The work was conducted under contract DASW01-94-C-0054, Task T-S5-1266, for the Defense Information Systems Agency. The publication of this IDA document does not indicate endorsement by the Department of Defense, nor should the contents be construed as reflecting the official position of that Agency.

# PREFACE

This document was prepared by the Institute for Defense Analyses (IDA) under the task order, Object-Oriented Technology Implementation in the Department of Defense (DoD), in response to a task objective to develop strategies for the implementation of object-oriented technology (OOT) within specific information technology areas within the DoD. This document is one of a set of four reports on OOT implementation. The other reports, focusing on other areas of OOT, are IDA Paper P-3142, *An Object-Oriented Development Process for Department of Defense Information Systems*; IDA Paper P-3143, *Object-Oriented Programming Strategies for Ada*; and IDA Paper P-3144, *Legacy System Wrapping for Department of Defense Information System Modernization*. All of this work was sponsored by the Defense Information Systems Agency.

The following IDA research staff members were reviewers of this document: Dr. Edward A. Feustel, Dr. Richard J. Ivanetich, Dr. Reginald N. Meeson, Dr. Judy Popelas, Mr. Clyde G. Roby, and Mr. Glen R. White.

# Table of Contents

Table of Contents

# List of Figures

# List of Figures

# List of Tables

# EXECUTIVE SUMMARY

## Purpose

Systems reengineering is building a new system using an existing system as the basis for requirements or design. As an activity that encompasses a combination of other activities, systems reengineering has the goal of improving the software system in terms of functionality, performance, and/or implementation. This report describes a set of strategies using object-oriented technology (OOT) for reengineering information systems in the Department of Defense (DoD). It specifically addresses the transition from the process-oriented business and systems analysis models used in legacy systems to the use of object-oriented (OO) analysis models. The audience of this report includes DoD software development managers, project managers, technical leads, and software engineers.

## Transition Issues

Two issues were identified as critical in comprehending the effort required for an OO-based systems reengineering effort.

- *Non-OO specifications in legacy systems.* Artifacts (requirements, design, and database specifications) obtained from a legacy system will probably not be in an OO form. The majority of older systems were built before either OOT or even structured techniques were in common use. Consequently, it may not be a simple issue to forward engineer a new system using these non-OO specifications.
- *Non-OO functional process improvement policy.* Current DoD policy requires that a functional process improvement activity be carried out before reengineering an information system, but the techniques and models to support functional process improvement are not object oriented. These functional process improvement models are supposed to be used as input to the information system reengineering or development.

To help address these issues, this document focuses on transitioning extracted requirements and design products that are not object oriented to an OO form. The transition of modeling paradigms is a relatively unexplored area but has become more important with the

reengineering of functionally based systems to OO form. The strategies presented in this document do not constitute formal mappings between original model types (IDEF0, IDEF1X, structured analysis) and the resulting model types (use case/scenarios, OO analysis, OO design), but are ideas for extracting objects with the goal of creating OO specifications.

In order to effect system reengineering based on OO analysis models, a paradigm shift is required from either the system artifacts or functional process improvement models. While strategies and guidelines are given here on using these extant models, the paradigm shift will not be automatic. The building of an OO system will still require substantial effort by system developers to construct OO specifications and models.

**Software Reengineering Example**

To help illustrate some of the specific considerations that must be addressed, this paper includes a detailed example of a legacy system that was reengineered to use OOT. This system is part of the Base-Level System Modernization (BLSM) program at Gunter Air Force Base, Alabama. BLSM has been designed to modernize base-level automated information systems (AISs) at U.S. Air Force bases and illustrates some of the reengineering strategies described in this report. The low level of interoperability and the difficulties in maintaining the current standard base-level AISs are principal drivers of the BLSM program. The software has become more difficult and expensive to maintain because of the magnitude of the code changes made over time. This situation is characteristic of many DoD legacy systems that face major modernization challenges.

One of the BLSM application systems—the Air Force Operations Resource Management System (AFORMS)—serves as an example of BLSM's approach to full-scale software reengineering using OOT. AFORMS is a legacy system that provided operations management information to operations supervisors to support effectively the implementation of Air Force flight management policies.

The presentation of the BLSM approach in this document outlines the general BLSM methodology and illustrates the application of that methodology to AFORMS, with a focus on the OO aspects of this reengineering program. Discussions are centered on the enterprise analysis for the entire BLSM program, an overview of the general BLSM software engineering environment, and the structure of individual software development phases for specific applications. These phases are illustrated with examples of OOT usage from AFORMS.

# CHAPTER 1. INTRODUCTION

## 1.1 PURPOSE

This document presents strategies for using object-oriented technology (OOT) when reengineering information systems in the Department of Defense (DoD). These systems include systems for command, control, communications, computers, and intelligence such as the Global Command and Control System, as well as information management systems for areas such as administration, personnel, supply, and maintenance. The document specifically addresses the transition from the process-oriented business and systems analysis models used in legacy systems to the use of object-oriented (OO) analysis models. Risks, problems, and issues are identified when using OOT in DoD information systems. The document does not discuss issues related specifically to legacy system wrapping, nor does it provide a review of OO analysis and design techniques for new systems development.

To illustrate the transition issues related to reengineering, an example of a software reengineering project is discussed in detail, the Base-Level Systems Modernization (BLSM) from the United States Air Force Standard Systems Center, Maxwell Air Force Base, Gunter Annex, Alabama.

The audience of this report includes DoD software development managers, project managers, technical leads, and software engineers.

## 1.2 BACKGROUND

Systems reengineering is essentially building a new system using an existing system as the basis for requirements or design. It is an activity that encompasses a combination of other activities such as reverse engineering, forward engineering, restructuring, redocumentation, and code translation, with the goal to improve the software system in terms of functionality, performance, and/or implementation [CIM94].

When reengineering a legacy system, the use of OOT introduces certain complexities into the software analysis, design, and implementation. First, the software system was not originally designed and implemented using an object-based approach. The products of reverse

engineering, such as requirements or design specifications, will probably reflect a functionally based approach. As a result, some degree of "transformation" of analysis and design models will be necessary in order to use those specifications. Second, the current policy in the DoD regarding information system reengineering requires that a functional (business) process improvement activity precede the systems reengineering activity [DOD92]. DoD Directive 8020.1 specifically requires the use of two analysis techniques, IDEF0 and IDEF1X, for functional process modeling. Neither one is object oriented, and their use can result in an awkward transition to systems using OO analysis.

It should be noted that while reengineering is applied to a legacy system, it differs from the wrapping strategies discussed in a companion document [IDA95b]. System wrapping isolates the legacy system or system components, but does not dramatically alter the existing system. Reengineering, on the other hand, can alter the code, design, and requirements of the system. The new system may need to reflect a new functional process, or it may reflect the old functional process but have additional system requirements. In other cases, the system requirements maybe sufficient, but the system requires a new design or an improved implementation.

## 1.3 ORGANIZATION OF DOCUMENT

Chapter 2 discusses the transition from a non-object-oriented environment to an object-oriented systems development, with the primary discussion focusing on transitioning extracted requirements and design products that are not object oriented. Details are provided concerning what reengineering involves, what decisions are required during the reengineering process, the determination of project scope, and the type of software elements to extract during reverse engineering.

Chapter 3 describes the BLSM effort, outlining the general BLSM methodology and illustrating the application of that methodology to AFORMS (Air Force Operations Resource Management System). It begins with discussion of an enterprise analysis for the entire BLSM program, proceeds to provide an overview of the general BLSM software engineering environment, and then focuses on structure of individual software development phases for specific applications. These phases are illustrated with examples of OOT usage from AFORMS.

Chapter 4 summarizes the guidelines and issues regarding reengineering.

References, glossary, and a list of acronyms are provided at the end of the report.

# CHAPTER 2. TRANSITION APPROACHES FOR REENGINEERING SYSTEMS

This chapter discusses aspects of systems reengineering that are of concern when transitioning to an OO implementation. Primarily we focus the discussion on transitioning extracted requirements and design products that are not object oriented to an OO form. There is background discussion concerning what reengineering involves, what decisions are required during the reengineering process, the determination of project scope, and the type of software elements to extract during reverse engineering. It should be noted that the transition of modeling paradigms is a relatively unexplored area but has become more important with the reengineering of functionally based systems to OO form. The strategies presented here do not constitute formal mappings between models but are ideas for extracting objects with the goal of creating OO specifications. Specific model transitions are shown in Figure 1.

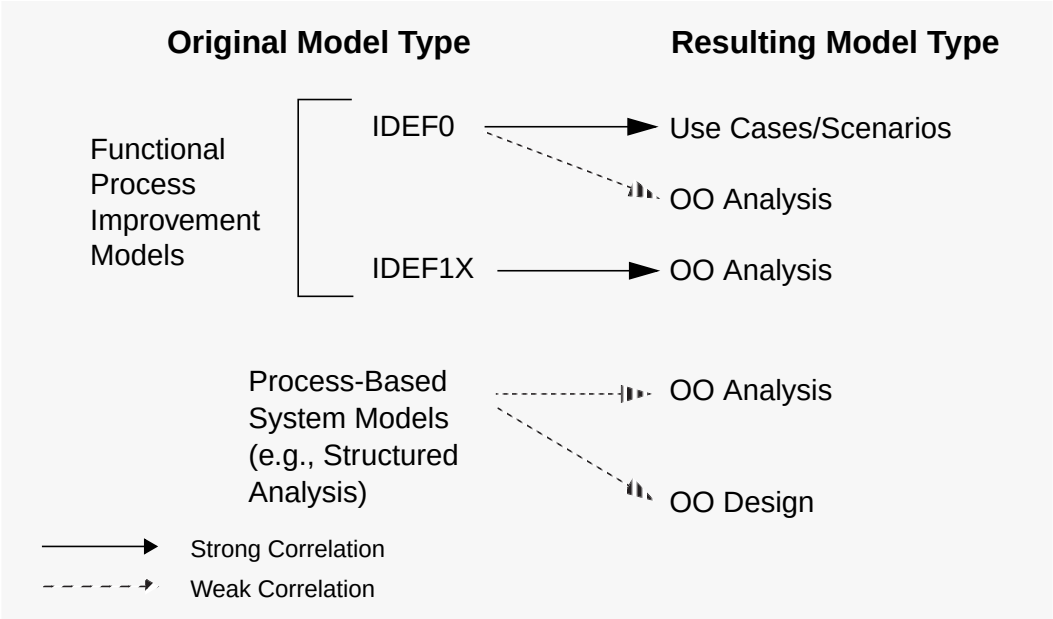


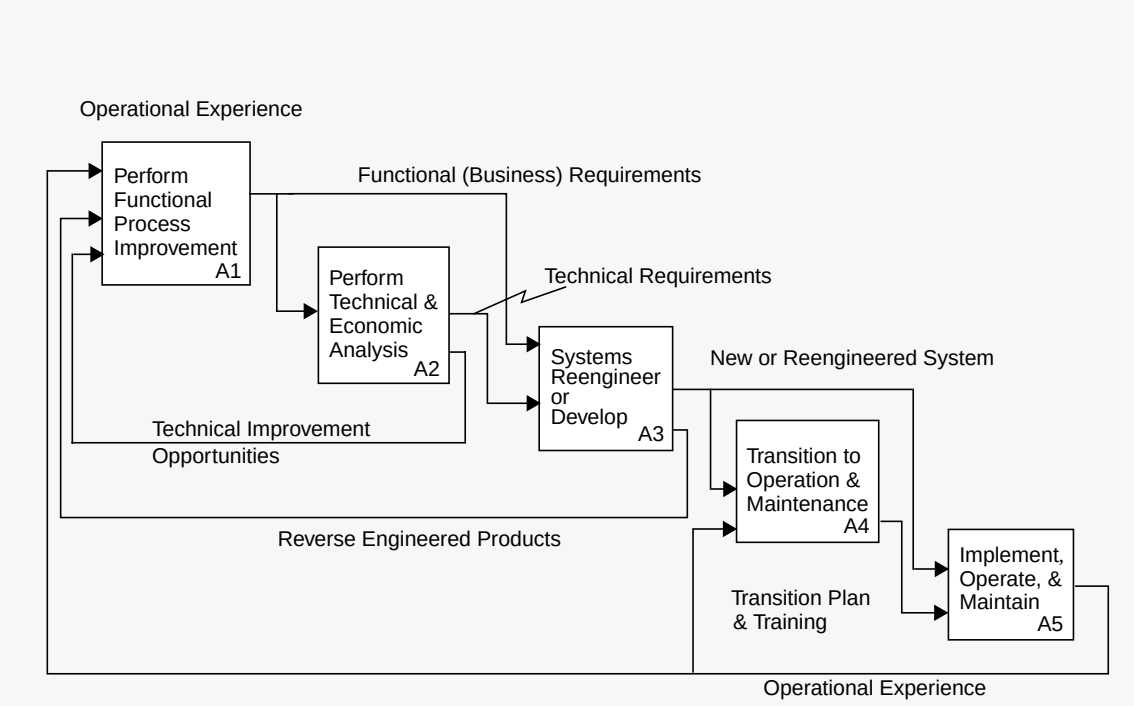

**Figure 1. Non-OO to OO Transitions**

Throughout this chapter, we use a hypothetical Aircraft Maintenance System (AMS) to illustrate transitions and mappings from non-OO to OO paradigms. Different aspects of the AMS are highlighted and discussed. Example object models are shown with Rumbaugh's Object Modeling Technique (OMT) notation [RUMB91].

## 2.1 REENGINEERING PROCESS

### 2.1.1 Functional Process Reengineering

Whenever an information system is to be reengineered, current DoD policy requires that a functional process improvement activity precede the systems reengineering [DOD92b, DOD93a]. This functional process improvement is the first activity within the DoD's Information Management Process as depicted in Figure 2.



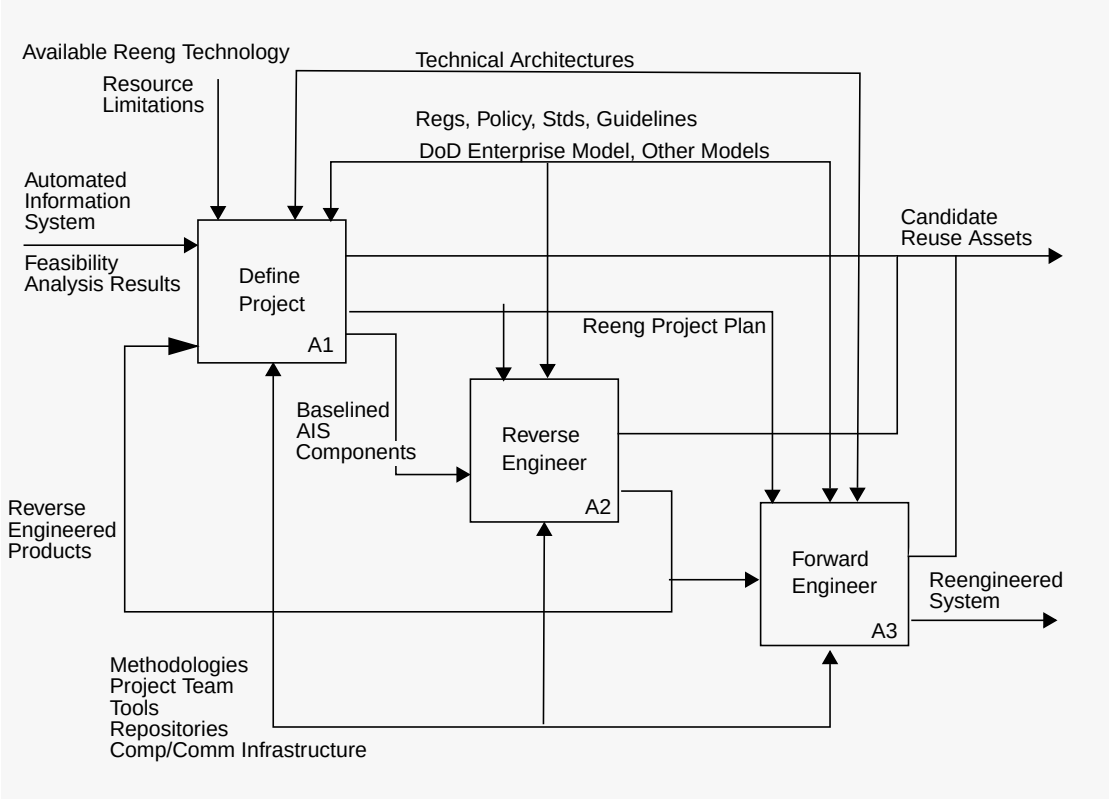

Source: [CIM94]

**Figure 2. Information Management Process Model**

DoD Directive 8020.1 specifically requires the use of two analysis techniques, IDEF0 and IDEF1X, for the modeling of an enterprise. IDEF0 is a technique and notation that models the *activities* of an enterprise; IDEF1X models its *entities*. The use of IDEF0, further discussed in Section 2.3.1 on page 10, supports the reorganization and revision of an enterprise's business activities.

### 2.1.2 Software Systems Reengineering

The Defense Information Systems Agency Center for Information Management (DISA/CIM) has defined a process for informations systems reengineering, depicted in Figure 3, in the CIM Software Systems Reengineering Process Model [CIM94]. This model divides a major reengineering effort into three major activities: (1) Define Project, (2) Reverse Engineer, and (3) Forward Engineer. Forward engineer encompasses activities associated with traditional developments, i.e., requirements analysis, design, implementation, test, and maintenance. The difference from traditional developments is that there is a base of existing products (the system and anything reverse engineered) upon which to proceed.




Source: [CIM94]

**Figure 3. CIM Software Systems Reengineering Process Model**

### 2.1.3 Scoping the Reengineering Effort

Since reengineering includes a variety of software engineering activities that range from simple code restructuring to full-scale reverse and forward engineering with new requirements, it will be necessary to consider the scope of the reengineering carefully. Examples of questions to consider include the following:

- Does the specification accurately (correctly and completely) represent the requirements of the new system requirements? In some cases of reengineering, there will be no change to system requirements and the reengineering can start with the design phase of development.
- Are there changes to the functional (business) process? It may be possible for the existing automated system to support a new functional process, but a new functional process will very likely impose new requirements. For example, there may be a different set of users and data sources.
- Are there new requirements for the system but no change in functional process. The requirement for better system performance, for example, often motivates the examination of existing system resources with resulting reengineering or upgrade efforts.

*It should be noted, however, that given the quality of the legacy system and its components, it may not be worth the effort to reengineer those components, which essentially leaves a new development activity. One should not assume that system reengineering is less costly than a new development, particularly given the tools now available. A careful comparative analysis of the costs, risks, and benefits of varying system options should be made before embarking upon a reengineering effort.*

### 2.1.4 Related Documents

The following references are a source for general guidance to the overall reengineering process:

[DOD92] Department of Defense, *Functional Management Process for Implementing the Information Management Program of the Department of Defense*, DoD 8020.1-M, draft, August 1992.

[CIM94] Center for Information Management, *Center for Information Management Software Systems Reengineering Process Model, Version 2.0*, draft, Defense Information Systems Agency, Joint Interoperability Engineering Organization, September 1994.

[CIM93a] Center for Information Management, *Automated Information Systems Software Reengineering Risks Taxonomy Report*, Defense Information Systems Agency, Joint Interoperability Engineering Organization, September 1993.

[CIM93b] Center for Information Management, *Information System Criteria for Applying Software Reengineering: Guidelines for Identifying Candidate Information Systems for Software Reengineering,* Defense Information Systems Agency, May 1993.

[IDA86] Institute for Defense Analyses, *A Descriptive Evaluation of Automated Software Cost-Estimation Models,* Alexandria, VA, October 1986.

[IDA95a] Institute for Defense Analyses, *System Reengineering Assessment Method,* IDA Paper P-2904, Alexandria, VA, January 1995.

[IDA93] Institute for Defense Analyses, *User's Manual for the Functional Economic Analysis Model (Version.3.0),* Alexandria, VA, December 1993.

[JLC93] Joint Logistics Commanders Joint Policy Coordinating Group on Computer Resources Management, "Reengineering Economics Handbook," *Proceedings of First Software Reengineering Workshop - Santa Barbara I,* March 1993.

[STS93a] Software Technology Support Center, *Reengineering Technology Report,* Hill Air Force Base, UT, August 1993.

[STS93b] Software Technology Support Center, *Software Estimation Technology Report,* Hill Air Force Base, UT, March 1993.

## 2.2 REVERSE ENGINEERING

Reverse engineering is an essential part of the reengineering process. DISA's CIM Software Systems Reengineering Process Model defines reverse engineering as "the process of examining an information system by analyzing its documentation, application software, and data structures within the environment in which the information system operates. This analysis is performed to (1) identify the system's components and their interrelationships, and (2) create representations of the system in another form or at a higher level of abstraction. The goal is to understand the existing software system (functional, performance, or implementation). Extracted information is represented in a format which can be integrated into the life cycle for development of a software system" [CIM94].

### 2.2.1 Artifacts and Products of Reverse Engineering

The reverse engineering activity can use a variety of legacy components (artifacts) to develop better representations of the existing system.These products have varying levels of usefulness to the forward engineering activity. In some cases, the existing specifications (artifacts) are sufficient to convey an understanding of the system. In other cases, specifications will need

to be developed from reverse engineering source code. The following are typical of the artifacts from reverse engineering.

- **Source code.** The source code is the formal representation of the system. In other words, legacy requirements and design specifications may be either informal or simply out of date. By examining the source code, it is possible to develop the design structures, data requirements, system requirements, functional requirements, and business rules. There are tools on the market that help with the reverse engineering of code, either by restructuring old code or by identifying module structures. Bruce [BRU92] provides some guidance on the extraction of data elements from Cobol code.
- **Data models.** This includes database schema, data element definitions, and data files. Since legacy systems are often information intensive, the extraction of database schemas, file structures, etc., is essential to any data reengineering. It is often the case that flat file structures of the legacy system will be reengineered into a relational form. The entities in the data model can also provide a good start to the development of an object model since these entities reflect the items in the problem domain on which we want to keep information.
- **Design specifications.** The design specifications (if up to date) can provide structural information for both preliminary and detailed design levels. However, the use of design specifications may have limited usefulness since module names may be cryptic and understandable only to the original developers.
- **Requirements specifications.** The requirements specifications (if up to date) can provide system and functional requirements. The system specifications, however, are not the implementation and may not accurately represent the current system. So some degree of caution should be used when evaluating design and requirements documentation. Design documentation, if used, should be verified against the code, and requirements documentation should be verified with current users.

### 2.2.2 Scoping Reverse Engineering

Deciding which elements to reverse engineer is important since this activity can be time consuming and labor intensive. Reverse engineering source code can provide design, requirements, business rules, and data specifications. An assessment of the reengineering project can help determine which elements need to be extracted. It may be the case that only the data requirements are necessary since a new functional process and application will be developed.

In this situation, it will probably not be necessary to reverse engineer the old application code. It should be noted that each situation is different, and the software engineers need to assess carefully those products they need to extract.

Reverse engineering also has its limitations, especially as a means to extract requirements. The existing code (on its own) may not provide an accurate representation since (1) system reengineering is often accompanied by functional (business) process reengineering, which will introduce new requirements; and (2) the existing code may not be fully exercised by the current group of functional users. As systems evolve over the years, they are upgraded, patched, and reworked, often leaving areas of non-executed code. This type of code may not be obvious to identify and even if the code can be executed, the function it supports may no longer be needed by the functional user. As a result, if there is any change to system requirements, it will be necessary to work with functional users to identify or validate requirements for the reengineered system.

### 2.2.3 Reverse Engineering Guidance

The following sources provide guidance and experience on reverse engineering:


[AIK94] P. Aiken et al., "DoD Legacy Systems—Reverse Engineering Data Requirements," *Communications of the ACM*, Vol. 37, No. 5, May 1994.

[ARN93] R. Arnold, ed., *Software Reengineering*, IEEE Computer Society Press, 1993.

[BRU92] T. A. Bruce, *Designing Quality Databases with IDEF1X Information Models*, Dorset House Publishers, New York, 1992.

[NIN94] J. Ning, A. Engberts, and W. Kozaczynski, "Automated Support for Legacy Code Understanding," *Communications of the ACM*, Vol. 37, No. 5, May 1994. pp. 50-57.

[PRE94] W. Premerlani and M. Blaha, "An Approach for Reverse Engineering of Relational Databases," *Communications of the ACM*, Vol. 37, No. 5, May 1994.

[RUG90] S. Rugaber, "Recognizing Design Decisions in Programs," *IEEE Software*, January 1990.


## 2.3 FUNCTIONAL PROCESS IMPROVEMENT MODELS

As noted earlier in Section 2.1.1, DoD Directive 8020.1 requires that a functional process improvement activity be carried out before any significant system reengineering. The intent of this directive is to verify that an outmoded functional process is not simply automated

since it is generally believed that improvements in the functional process will yield substantially higher benefits, in terms of cost savings, than upgrades or reengineering of the supporting automated systems. Therefore, the directive requires the use of two particular modeling techniques, IDEF0 and IDEF1X, before any system development or reengineering takes place. These techniques model the activities and entities of an enterprise, providing in one case (IDEF0) a functionally based view of the enterprise and in the other (IDEF1X) an entity-based view. It is the IDEF0 model that often provides the insights for process improvements and may be the only model available at the time when system analysis begins. The IDEF1X model, which is initially derived from the IDEF0, often serves as the basis for database design of the eventual information system. Neither of these models provides a true OO view of the enterprise, so the question arises as to how to use these artifacts of the functional process improvement activity when employing OO approaches in the system-level development.

### 2.3.1 IDEF0 Models

In general, little has been published regarding the transition of IDEF0 models to an OO form. In comparing IDEF0 to OO models, there are two basic differences. First, the semantics defined in each are substantially different: IDEF0 is activity based, while an OO analysis model is object based. Second, the perspective of each model will likely be different: IDEF0 will probably be broader in scope providing a view of the enterprise, while an OO analysis model will be more constrained to a system-level view.

In the following sections two general strategies are presented for using existing IDEF0 models when pursuing an OO system development. In the first strategy, the IDEF0 model serves as a basis for constructing use cases or scenarios (Section 2.3.1.1 and Section 2.3.1.2). In the second strategy, potential "objects" are extracted from the IDEF0 in a manner similar to constructing the IDEF1X model (Section 2.3.1.3). However, this particular strategy is rather *ad hoc* and leaves substantial work to achieve a finished object or class model. An example transition, using an information system that manages an inventory of aircraft parts, is given in Section 2.3.1.4.

#### 2.3.1.1 IDEF0 Notation

IDEF0 uses the paradigm of top-down functional decomposition to provide a picture of the enterprise. In this technique, depicted in Figure 4, an overall activity is decomposed into a small set (three to six) of subactivities that are needed to realize the behavior of the original activity, and the decomposition can be continued for as many levels as necessary. The activities are affected by inputs, controls, and mechanisms, and produce outputs. An activity is represent-

ed with a box, and the inputs, controls, outputs, and mechanisms (ICOMs) are represented with arrows. The mechanism, which may be a person or device that carries out a function, or even a "tool" (hardware or software) that is used in the process, is the least important, particularly at the modeling levels most often used, and it is often omitted. But every box will have a control arrow that specifies the conditions under which the activity is invoked. This is often the output of another activity. The input is the item that is transformed, and the output is result of the activity. The arrows in IDEF0 are not thought of as sequences (as in a traditional flowchart) but as availability (of data or other items needed for the activity to work). Any sequencing is implicit, based on the availability of inputs, controls, and mechanisms.

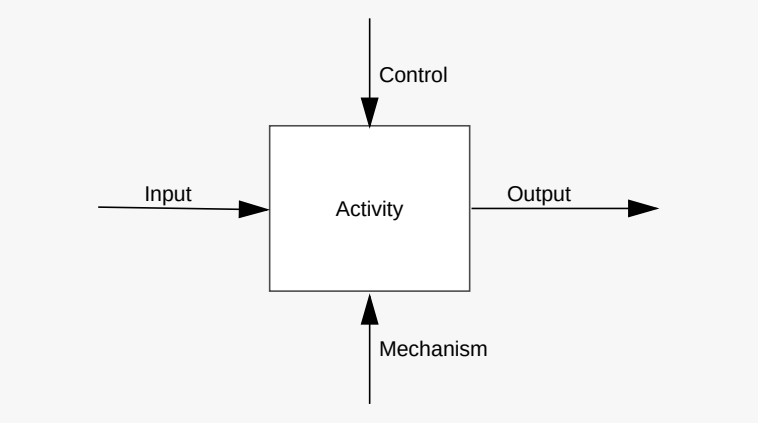



**Figure 4. IDEF0 Notation**

#### 2.3.1.2 Use Case or Scenario Transitions

Since IDEF0 models depict the major activities of the enterprise, they can serve as a starting point for use cases or scenario development. Use cases or scenarios are step-by-step descriptions of the process or event sequences of a system or enterprise. Use cases and scenarios are advocated by Jacobson [JACO93], Booch [BOO94], and Rumbaugh [RUMB91] as part of OO analysis. Jacobson and Booch scenarios are developed prior to constructing an object model, while Rumbaugh scenarios support the construction of interaction diagrams during dynamic modeling.

Depending upon the level of detail in the IDEF0 model, each activity may be represented as a distinct use case. If the activities are broken into smaller subactivities, then these subactivities may better correspond to the specific steps within an interaction diagram or scenario script. It is important to note that this transition will not be automatic and requires some degree of judgment in developing scenario details.

#### 2.3.1.3 Extraction of Objects

But suppose one is starting from an existing IDEF0 model and wants to use the system information in that model to move to an object model or to supplement the model with objects? The original model may have tried to avoid the mention of entities in function names. Here the suggestions of Ruegsegger [RUE93] can be used.

Ruegsegger's method produces the objects in a fairly straightforward manner. It is based on the observation that most IDEF0 arrows consist of either objects in the problem space, object attributes in the problem space, or object aggregates in the problem space, and that the mechanisms correspond to objects in the solution space. Thus, if the labels on the arrows are gathered, one gets the set of candidate objects.

The trick, of course, is to gather the labels at the appropriate level of granularity. If objects that are obtained which are not intuitive, in terms of either their initial appearance to the modeler or an organization expert working with the modeler, then it is important to move a level deeper. Likewise, if it is not possible to formulate the desired methods to go with the objects without adding additional constructs, this will force a deeper examination of the original IDEF0 decomposition as well.

The next step is to check for potential objects that are redundant (i.e., two names for the same thing). Ideally, these redundancies should have been resolved in the original modeling process, but this is a common flaw in modeling. Others redundancies will be attributes or relationships. Those that remain are the candidate objects, placed in an IDEF1X-like framework with the relationships labeled. This is circulated to various expert reviewers to get a consensus view of the model. If no consensus is forthcoming, then the modeler must decide the arrangement of objects and data flows.

It should be remembered that a group of people probably will not have a common view on the importance or arrangement of objects and data in a model. Different people are likely to have different models of the world, based on their experiences, and they adapt their internal models of any new systems they meet to the models with which they are familiar. There are stylistic differences in the way people think, like active and passive views, nominal and verbal modes of expression. But if one can attempt to reach some consensus, then even those who have different models in their minds will usually come to be able to understand the system in terms of the models developed.

#### 2.3.1.4 Transition Example: IDEF0 to Scenarios

The example is an information system that manages an inventory of aircraft parts for an Aircraft Maintenance Facility (AMF). Organizations that require aircraft parts place a parts order with the AMF, which in turn processes the order, extracts the requested part, and sends it to the requesting organization. If the AMF has depleted its supply of a particular part, then a request will be made to the part supplier to replenish its supply. To support the management of its parts inventory, the AMF plans to reengineer its primary information system, known as the Aircraft Maintenance System (AMS). The reengineered AMS will process new shipments, send needed parts to requesting organizations, and issue new shipping requests when inventory levels drop below a preset threshold.

An example of an IDEF0 context diagram is shown in Figure 5. The AMS is the main activity, with Parts Order and New Shipment as inputs; Part Information as a control; Receiving Clerk, Shipping Clerk, and Inventory Administrator as mechanisms; and Shipping Requests and Aircraft Parts as outputs.

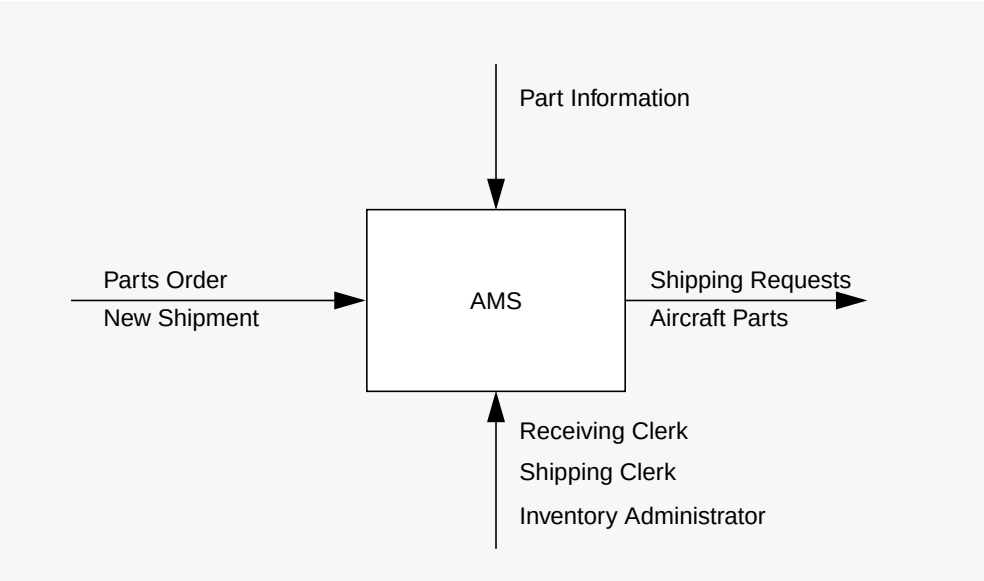



**Figure 5. AMS Context Diagram in IDEF0**

If we decompose the AMS activity, depicted in Figure 6 on page 14, we see that it comprises three subactivities: Process New Shipment, Send Aircraft Parts, and Order New Parts. A New Shipment comes into the AMS and is processed by the Receiving Clerk resulting in an Augmented Inventory. A Parts Order is handled by the Shipping Clerk to send out requested

Aircraft Parts. Last, the Inventory Administrator periodically reviews the inventory and issues a Shipping Request when stock levels fall below a set threshold. Although these activities are depicted in a left to right manner, there is not an explicit sequence between these activities, such as "do A1, then A2, then A3." As noted previously in Section 2.3.1.1, any sequence is the result of the availability of inputs, controls, and mechanisms.

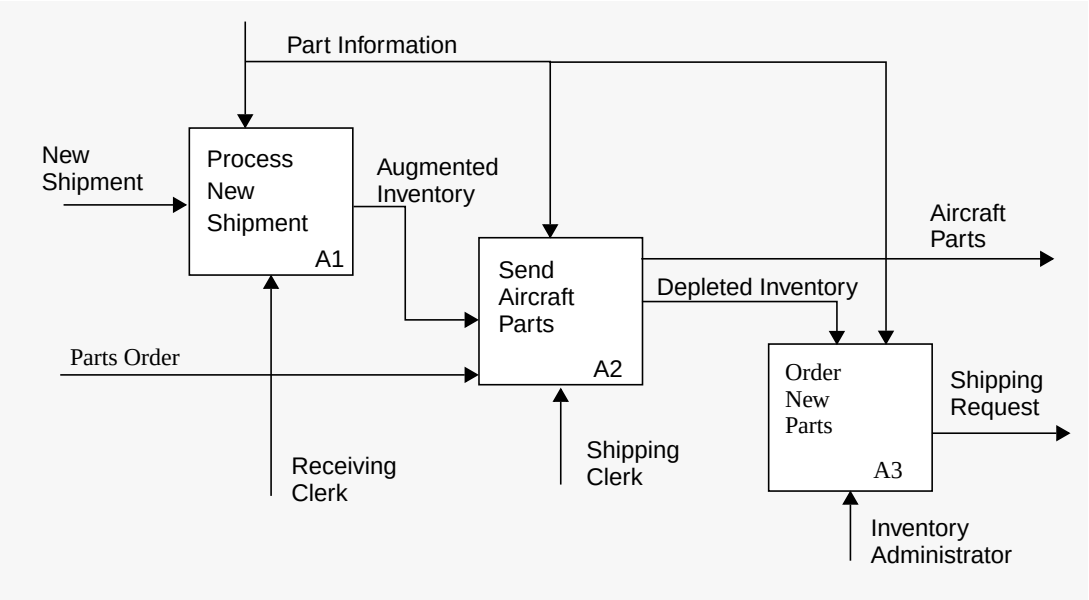



**Figure 6. AMS Top-Level IDEF0 Diagram**

In Figure 7 on page 15, the IDEF0 activities are represented as like-named scenarios of the AMS. Each scenario has specific steps, and each scenario is associated with a specific actor or type of user. The Receiving Clerk is the actor for Process New Shipment, the Shipping Clerk the actor for Send New Parts, and Inventory Administrator for Order New Parts.

### 2.3.2 IDEF1X Models

Databases have generally been developed by the same processes as software: analysis followed by design and implementation. But for more than a decade now, there has been a shift in emphasis from function-centered design to data-centered design, and the analysis has shifted correspondingly from problem centered to data centered. The reasons for this change in perspective are many, but the strongest has probably been that databases are now so common that they are used everywhere in organizations, and since data elements tend to be common over different parts of the organization, it does not make sense to have a single-purpose database. Thus, it makes sense to model the data once and then consider all the functions on it—and to

1. Process New Shipment.
   - Input shipment code.
   - Input item number and count.
   - Update inventory.

2. Send Aircraft Parts.
   - Scan Parts Order for part number, quantity, and destination.
   - Retrieve requested parts.
   - Package and label parts for shipping.

3. Order New Parts.
   - Check inventory level.
   - Identify needed items and quantity.
   - Identify lowest cost supplier.
   - Print Shipping Request.

**Figure 7. AMS Candidate Scenarios**

be able to add more functions as needed—than to build the database system about functions. On the corporate information management level, the use of common data elements cuts duplication of effort and aids in departmental interchange of data, implementation of common software, and communication with extramural organizations.

Preceding this data-centered approach, but very much serving as a catalyst in bringing it about, is the realization that if data is to be multi-use, it needs to be handled at a level that abstracts from both its physical structure and any particular use to which it might be put. This is the notion of a *conceptual schema* for data in a database, or as it is often called, a *conceptual model* for an organization. (Synonyms are *semantic data model* and *information model*.)

IDEF1X is a conceptual modeling notation and technique based on the entity-relationship (E-R) approach that allows the development of databases from the conceptual model. IDEF1X and similar modeling techniques allow more than the development of databases. Conceptual models of entities, attributes, and relations are vital parts of detailed enterprise models since the types of information handled in an organization tell a lot about the functioning of that organization. IDEF1X is organized around the entities on which information is being kept; therefore, it is not so terribly far from the OO view. In this section, we will discuss how one can extract from IDEF1X models the information that can be

used in forming OO models. We will also discuss the limitation of doing this: what information may be missed and where one needs to look for that information.

Although IDEF1X can be used to define database schemas, it can be extended to provide models of anything in the domain being modeled that is an entity and has attributes and relationships to other entities. This can include animate or inanimate objects, physical objects or virtual objects, and/or textual or non-textual media objects.

In an enterprise model, it is advantageous to include anything that is considered an important entity in the processes of the organization. There are some entities, including organizational entities such as departments that people manage, that may be irrelevant to the processes, and if so, they are not needed in the models (and probably not in the enterprise, either, if one is trying to develop the most effective enterprise to get the job done). For this reason, it is useful to consider processes in defining entities. More will be said in the next section about the need to alternate between analyzing processes and analyzing entities.

#### 2.3.2.1 IDEF1X Notation

In an IDEF1X model, depicted in Figure 8, the basic unit around which data is organized is an *entity.* From that point of view, the model is very much like an object model, which provides some promise that a mapping can be found. It should be noted that the description here is a quick overview of IDEF1X. The IDEF1X notation has an extensive set of rules for its use. References such as [BRU92] should be consulted for complete details on IDEF1X syntax and semantics.

The following are definitions for the basic components of the IDEF1X notation [BRU92]:

- Entity. “Any distinguishable person, place, thing, event, or concept about which information is kept . . . An entity is represented by a closed box with the name of the entity at the top and the attributes of the entity listed inside the box.”
- Independent entity. “An entity that does not depend on any other for its identification . . . Independent entities are represented by square-cornered boxes.”
- Dependent entity. “An entity that depends on one or more other entities for its identification . . . Dependent entities are represented by boxes with rounded corners.”
- Associative entity. “An entity that inherits its primary key from two or more other entities (those that are associated) . . . Associative entities record multiple associations (relationships) between two or more entities.”

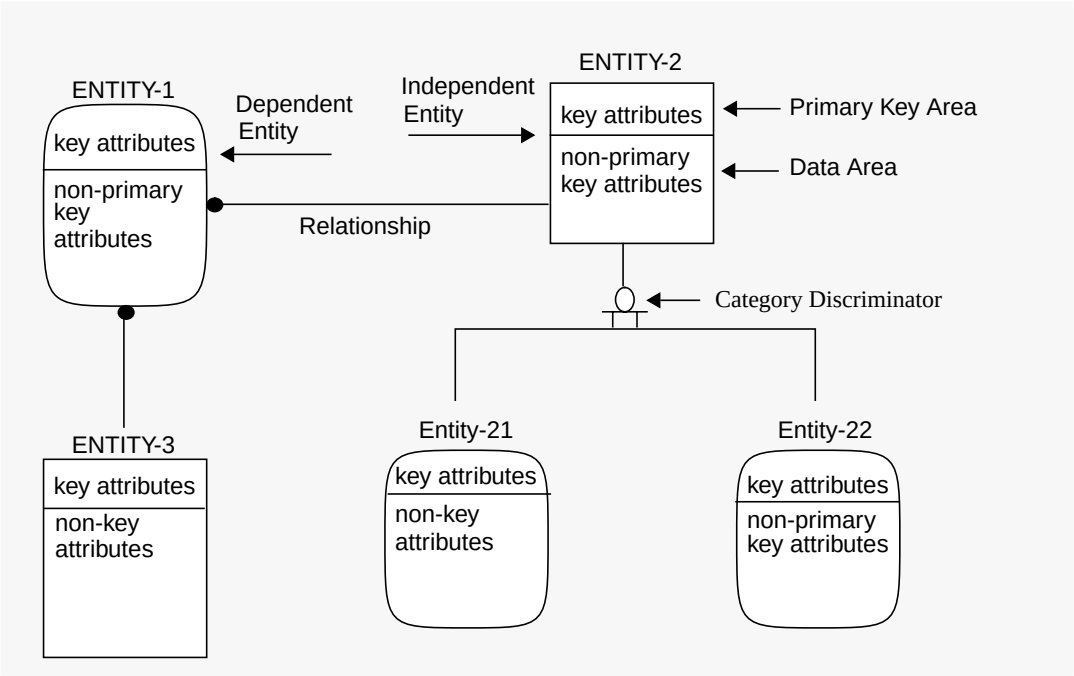



**Figure 8. IDEF1X Notation**

- Instance. “A single occurrence of an entity.”
- Attribute. “A property of an entity.”
- Primary key: “An attribute or group of attributes that has been chosen as the unique identifier of the entity.”
- Primary key attributes. “An attribute that, either by itself or in combination with other primary key attributes, will form the primary key.”
- Non-key attributes. “An attribute that has not been chosen as a part of the primary key of the entity.”
- Relationship. “A connection between two entities.”
- Foreign key. “A primary key of an entity that is contributed to another entity across a relationship.”
- Category discriminator. “An attribute that determines to which category a generic parent instance belongs.”

#### 2.3.2.2 IDEF1X to Object Model Mapping

The first thing to be aware of in transitioning from IDEF1X to an object model is the point discussed previously in Section 2.3.2.1. The IDEF1X model is oriented to data, and if it has been produced for database purposes, the entity is an item about which data is kept in a store. It may not even be all of the data that would be in a data entity which is intended to be kept in a database, which may not cover all of the data in a full enterprise model. Furthermore, if there are non-informational entities to be included in the model, a model developed for database purposes will not have it. So if the object model is to be a fairly broad enterprise model, the IDEF1X models of the same enterprise may be much narrower and may only help with parts of the object model. Since IDEF1X models can be used for broader entity relation models which include non-informational data, this will depend on the particular model, who did it, and what was its purpose.

The perspective of object orientation is that the object provides the means of accessing data about itself. This is not such a leap from IDEF1X, merely a reinterpretation. Another subtle change between IDEF1X and object model has to do with the interpretation of the attributes. They are not thought of as passive data lying in a repository, necessarily. The object model allows them to be calculated or retrieved in any way. It is not necessary to specify the details of how the attributes are retrieved or calculated. Again, this is not a major shift in what can be described but a shift of perspective.

In general, there is a fairly direct mapping going from the IDEF1X model and to an object model. Although there are some terminology differences, the object model appears to be a superset of the IDEF1X concepts. Table 1 provides a summary of the mappings from IDEF1X to an object model.

**Table 1. IDEF1X to Object Model Mapping**

| IDEF1X Model | Object Model |
|---|---|
| Entity | Class |
| Attribute (non-foreign key) | Attribute |
| Relationship | Same |
| Cardinality | Same |
| Generalization (category discriminator) | Inheritance (without foreign keys) |
| Instance (Relation Tuple) | Class Instance (object) |

#### 2.3.2.3 Transition Example: IDEF1X to Object Model

In Figure 9, there are five entities shown for the hypothetical AMS: four are independent entities (Part, Supplier, Order, and Organization), and Supplied-part is an associative entity. Since Part and Supplier can have a many-to-many relationship, Supplier-part is added as an associative entity.

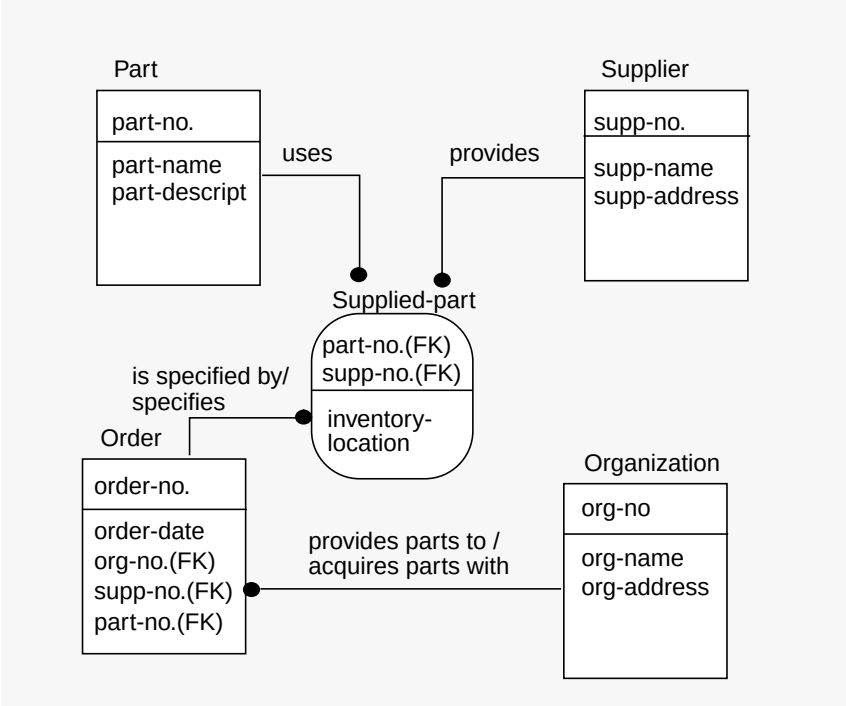



**Figure 9. AMS in IDEF1X**

In the object model, depicted in Figure 10, the IDEF1X entities, Part, Supplier, Order, and Organization map to classes. The associative entity, Supplied-part, maps to the class Supplied-part, which has a ternary relationship with Order where Order requires a specific part from a specific supplier. Generally, attributes that are foreign keys in the IDEF1X model do not need to be represented in the object model. Relationships in IDEF1X, including cardinalities and generalization, can map fairly directly to an object model; however, some relationships, such as the “uses” between Part and Supplied-part, may be eliminated or revised with the removal of associative entities. It should be noted that the resulting object model is only a partial model since no services or operations are identified.

## 2.4 PROCESS-BASED SYSTEM MODELS

The reengineering of an existing system may require the revision and transition of specifications that were written using non-OO techniques, such as Structured Analysis (SA) or its

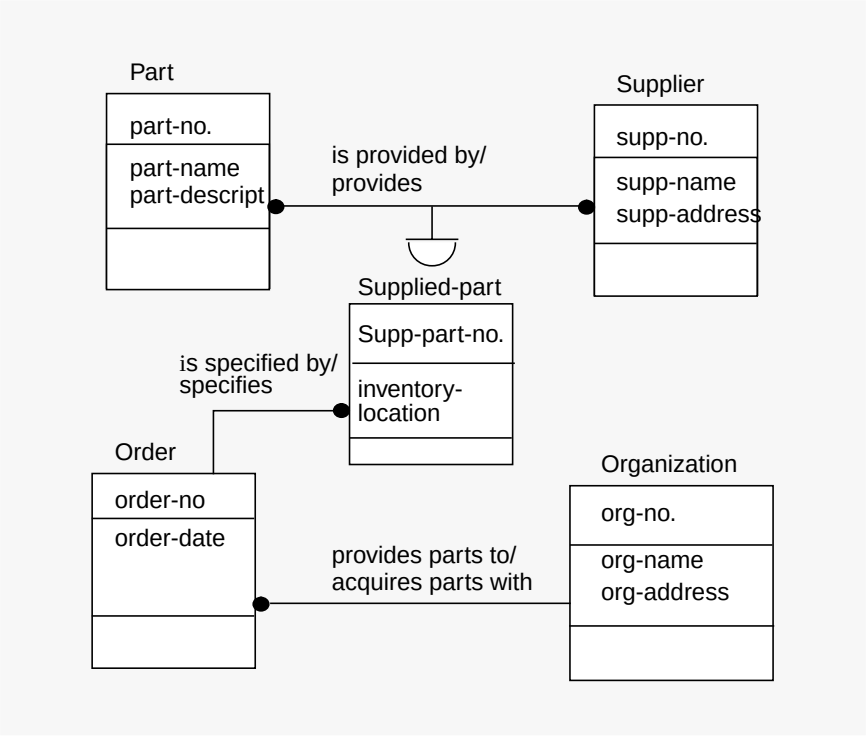



**Figure 10. AMS in Object Orientation**

successor, Modern Structured Analysis (MSA). Many of these "older" techniques were process based which allowed the distribution of data throughout the specification as well as the eventual program. This section examines those issues encountered when the forward engineering encompasses the requirements analysis and design stage. In this case, the options will be to transition an existing (or reconstructed) specification to an OO form or to create a new OO specification that will not have a close dependence upon the previous requirements specification.

### 2.4.1 Transitions to OO Analysis

SA and MSA are software analysis methodologies that partition a system into communicating asynchronous processes. This approach originated with DeMarco [DEM79] and was popularized by Yourdon's SA/MSA [YOUR89] and other analysis techniques such as the Hatley-Pirbhai [HAT87] real-time extensions to data flow diagrams. For the purposes of simplicity, we will refer to this general class of models as SA models since their semantics are similar.

#### 2.4.1.1 Structured Analysis Notation

SA notation, depicted in Figure 11, uses the data flow diagram (DFD) as the primary graphical notation, although it is supplemented with entity-relation and state transition diagrams. In this notation, the process is represented by a circle, data flow is represented by directed arcs, data stores are represented by two horizontal lines, and external sources and sinks (terminators) are represented by boxes. Typically, the terminators, which represent outside users or systems, are only shown on top-level context diagrams. The SA process can be decomposed into more detailed diagrams, containing processes, data flow, and data stores. In other words, the SA process can represent an entire system or subsystem and more than just a simple function.

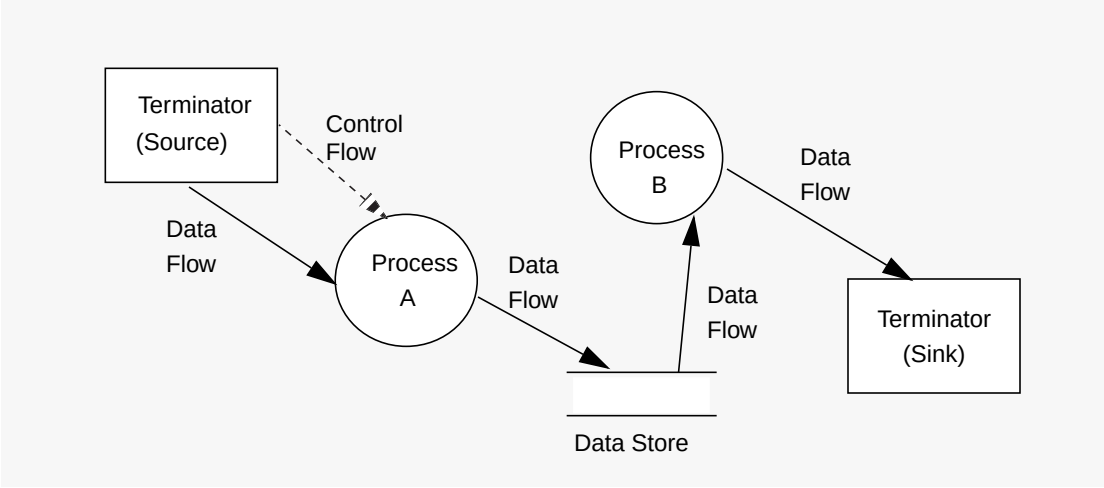



**Figure 11. Structured Analysis Notation**

It should be noted that although this technique is process based and looks similar to the IDEF0 discussed in the previous section, there are some differences. First, SA often represents a more detailed level of abstraction than the IDEF0. With SA, we are often modeling the information system exclusively, whereas IDEF0 is employed for modeling the enterprise at large. Second, the ICOMs in IDEF0 do not represent data flow and hence do not have a direct correspondence to the data flow arcs in SA. Third, SA does not distinguish between data that are transformed (such as an IDEF0 input) and data that are used but not transformed (an IDEF0 control). Last, the IDEF0 notation does not specifically identify data stores as does structured analysis.

#### 2.4.1.2 Structured Analysis to OO Analysis Mapping

It is possible to construct some semblance of an object model given an SA representation. This object model, however, will very likely be a partial model, with incomplete class or object definitions. The primary strength of using a DFD at this level is that the SA model uses the vocabulary of the domain. That is, the SA model attempts to model the domain from the user's point of view, using language that the user can understand as opposed to a design model of the system that depicts system modules, etc., using system-specific names that are not understandable to a user. This is important because we will depend upon that "domain vocabulary" represented in the SA model to assess the type of item being represented. Following is a summary of the mapping of the SA model to an object model.

- Terminators will generally map to a class or object in the problem domain. While terminators show up as class within a generalized object model, there may not be the need to represent such a class within the design.
- Data stores will map to a class or objects. Data stores generally represent passive objects within the system solution.
- Data flows can correspond to classes, objects, or attributes. Data flows can also represent attribute values. So it will be necessary to assess each data flow carefully for correspondence to an object model.
- Control flows often correspond to specific events since they represent some sort of signal to a process such as "start" or "stop."
- Processes can correspond to services or operations within a class. However, this correlation should be made very carefully since an SA process could be assocIated with more than one class.

#### 2.4.1.3 Bailin's Approach

Another method for developing an object model from a process model is Bailin's [BAI89], which was developed as a method of doing the original analysis of a system. This method, known as Object-Oriented (Requirements) Specification (OOS), works with data flow models and with E-R models. It produces an E-R diagram and a hierarchy of entity-dativa1 (not function-data flow diagrams, which together constitute an object model). The major practical weakness of his method is in the selection of objects, which is insufficiently well-defined.

Bailin writes that his method is not really new, that "under the guise of structured analysis, engineers have been doing OO specifications for years . . . Analysts have tacitly chosen to

specify and decompose entities when appropriate, disguising them as structured analysis processes." Bailin believes that decomposition in terms of processes is not a loss if one wants to find objects. If one already has SA models, and these are understood by analysts in the organization, then it may make sense to start with those.

**Step 1** of Bailin's OOS is, then, to produce a sort of normalized representation. The diagrams will have objects appearing in process names, in phrases of the form of an *action-object pair*, with a few extraneous words allowed, as long as the main pair appears. These objects will tend to be in the problem or enterprise domain.

**Step 2** of OOS is to distinguish between what Bailin terms "active entities" and "passive entities," according to the following requirements:

- Active entities should appear as processes, passive entities as data flow.
- Every function must be performed by some entity.

The Bailin criteria for active entities are less stringent than those for active objects. When OOS is being used for software development, the rule of thumb is that an active entity is one in the problem domain that will be considered in the design phase (because it contains functions that must be specified in that phase) and a passive entity is one in the implementation domain that will be considered later in the design phase.

**Step 3** depicts a top-level data flow diagram created with the active entities as objects (in boxes) and the passive entities assigned to stores or to data flows. An E-R diagram is developed for the relationships embodied in the flow diagram. Bailin gives some rules for maintaining consistency between object data flow diagrams and E-R diagrams:

- The entities in the two diagrams must correspond exactly.
- Any relationship in the E-R model must be manifested somehow, either (1) through containment, (2) as a data flow that is a passive entity being related to an active entity that it passes into or out of, or (3) as two active entities connected by a data flow.

He admits there may be some exceptions to this rule, in which relations are perceived to occur but cannot be shown.

**Step 4** is to decompose the entities or functions, using the following rules:

- Entities are decomposed into sub-entities and/or functions.
- Functions can only be decomposed into subfunctions.

Finding the functions of entities is just determining what they do, and these may be reflected in the original SA diagrams. If, in the original analysis of Step 1, there were action-object pairs, then the action is likely to appear here as a function of the object, or of some sub-object. One continues to refine both the data flow and E-R diagrams.

**Step 5** is to check whether new objects have been introduced as functions are decomposed. This question is asked repeatedly, and if the function is naturally expressed in terms of the action-object pair, the object should be in the diagrams or should be introduced. Passive entities may be left implicit if they are deemed too minor to introduce in the E-R model, but all new active entities must be shown.

As new entities are introduced, it is necessary to group the functions under the appropriate entities. This is done in **Step 6**, where an attempt is made to give a complete list of functions performed by or on the new entities. At the same time, old groupings of functions need to be examined to see if they more naturally fall under a new entity.

**Step 7** is to organize the entities into domains. To get a good object mode, we would develop "is-a" hierarchies and show where inheritances take place, which should be implicit in the E-R diagrams, what functions go with each entity, etc.

#### 2.4.1.4 Transition Example: Structured Analysis to Object Model

In Figure 12 is the AMS example starting with an SA context diagram. The AMS is shown as the primary process with New Shipment coming from a Supplier and Parts Order coming from user Organizations. As output, the AMS produces Shipment-Requests to the Supplier and Ordered-Parts to the user Organization. Both the Supplier and Organization are terminators in this model.

In the decomposition of the AMS, depicted in Figure 13, we identify a number of constituent processes within the AMS bubble. The data flow Parts-Order is input to the Delete Part process (part of Send Aircraft Parts), which in turn sends a specific Part-No. to the Parts-Inventory to be deleted. In a similar manner, New-Shipment is input to the Add-Part process (component of Process New Shipment), which in turn sends a specific Part-No. to Parts-Inventory to be added. One of the things to notice with DFDs is that, at this level, it is difficult to tell the level of complexity within a DFD process.

In developing a possible object model, depicted in Figure 14, both the context diagram and the AMS DFD are sources of possible objects or classes. While the data store Parts Inventory and the terminators Supplier and Organization are obvious as objects, the data flow

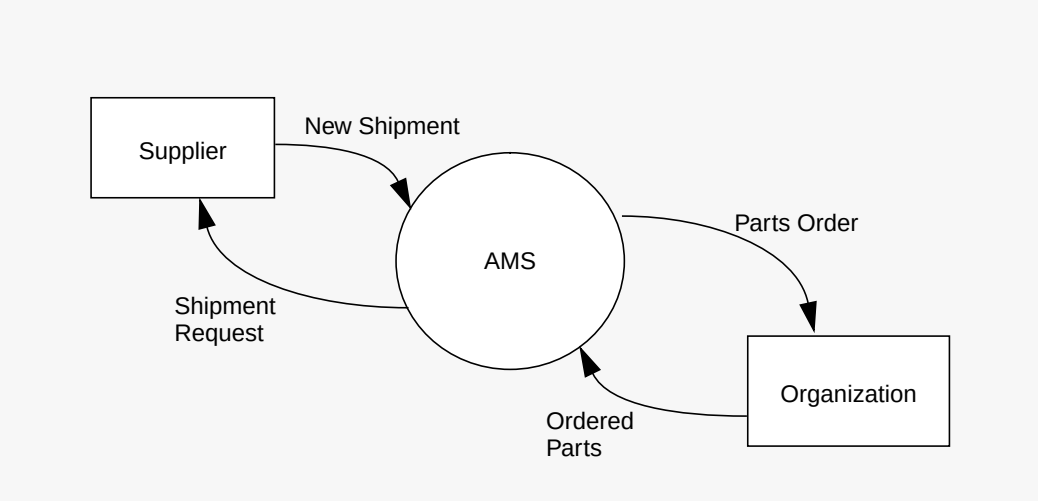



**Figure 12. AMS Context Diagram (Structured Analysis)**

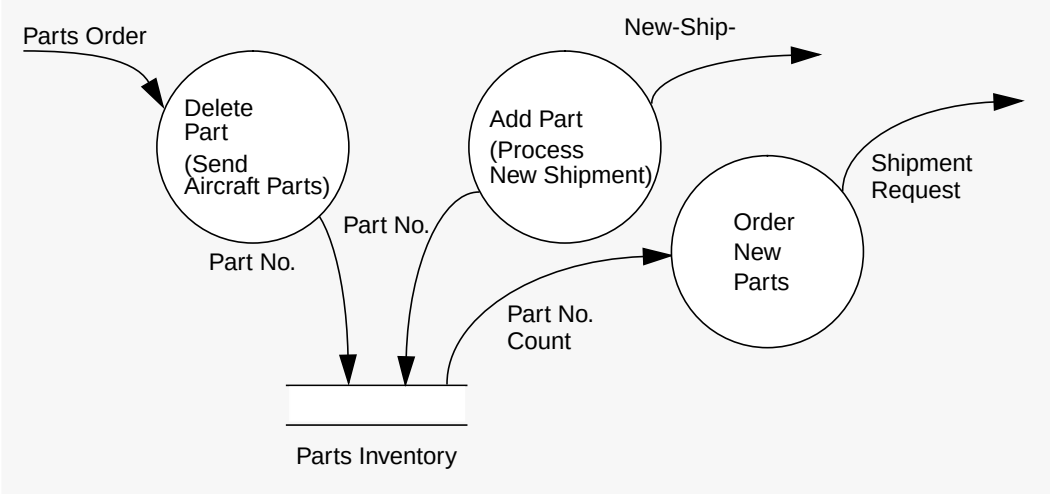



**Figure 13. AMS Data Flow Diagram (Structured Analysis)**

maps to possible objects (Parts Order, New Parts, Shipment Request) and to attributes (Part No., Count). The Parts object has the services Add-part and Delete-part and the attribute, Part-no. However, further interpretation will be needed to complete the object model. For instance, both Supplier and Organization will likely have attributes of name and address, but this is not represented in the context diagram. These attributes may be components of the Parts-Order and New-Shipment, possibly showing up in further development of the AMS DFDs. Some basic relationships are evident such as the send and receive relationships with Supplier and Organization.

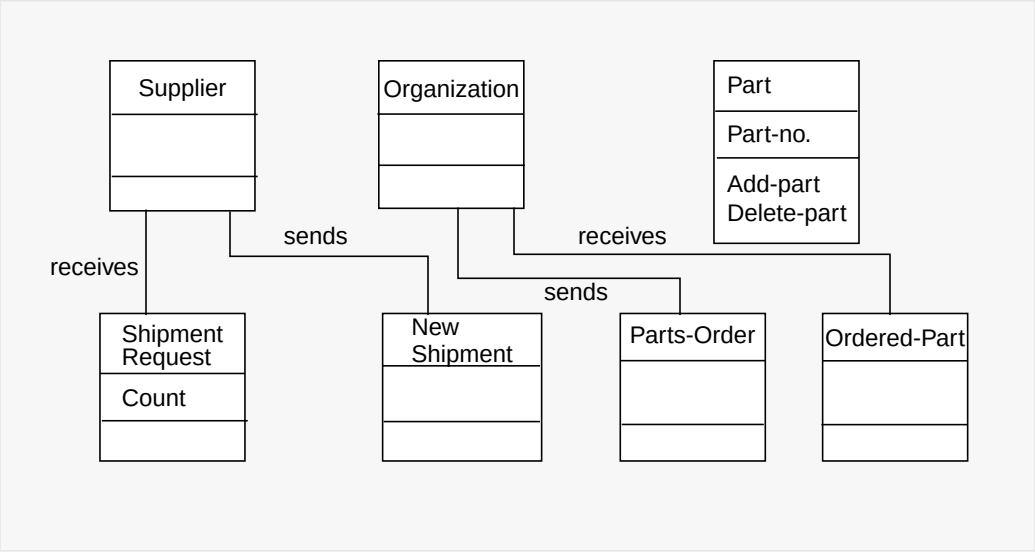



**Figure 14. AMS in Object Orientation (Partial Object Model)**

### 2.4.2 Transitions to OO Design

Before OO analysis techniques were developed, software developers were faced with transitioning from process-based analysis techniques, such as SA, to OO design. This transition presented a challenge since, as with the IDEF0 to OOA transition, there is a mismatch between modeling paradigms. Booch does not recommend using SA or process models with object modeling since this combination results in limited success [BOO94]. One of the problems encountered with going from SA to OO design is that the analysis picture of the system is partitioned along functional lines. SA processes may also be more complex than simple functions. SA processes can represent complex activities or subsystems that in themselves contain data stores or data flows. For further discussion on OO design and OO programming in Ada, see IDA95b.

#### 2.4.2.1 Structured Analysis to OO Design Transitions

The shift from SA to OO design requires two elements of transition. The first element of transition is from the process-based paradigm to OO paradigm, and the second element is from the requirements analysis phase to the design phase. Within the functional SA picture, potential objects may not be obvious or may be distributed throughout the system. New objects will also appear at the design phase. These objects are necessary to build the system but are not part of the application domain. In addition, it is also necessary to consider the implementation

language at the design phase. The example in the next section (Section 2.4.2.2 on page 28) presumes an Ada implementation, using Buhr diagrams to illustrates Ada constructs.

Nielson and Shumate [NIE88] define an approach for developing OO designs from SA specifications, called the Layered Virtual Machine/Object-Oriented Design (LVM/OOD). This approach is "based on the concepts of creating objects in layers of abstraction, information hiding, and stepwise refinement (deferring design decisions)" [NIE88]. In LVM/OOD, requirements specifications are first represented with DFDs, depicting major functional transforms and data flows. Nielson and Shumate then look for concurrent processes. They consider a basic process as an anonymous class and a concurrent process as an "object" which can be implemented as an Ada package, subprogram, or task. DFD processes are grouped along lines of concurrency. However, data abstraction is also employed, particularly when a process acts as a data manager or monitor around a data store. Since this approach is intended to support real-time system design, it is not always clear whether they want to emphasize data abstraction or concurrency as a means for grouping DFD processes. But this is one the few design approaches that transitions a specification from SA to OO design.

It may be possible to employ the principle of data abstraction primarily when evaluating a DFD. In that case, some of the following transitions may apply:

- **Terminators.** Although terminators will generally map to classes or objects in the problem domain, they may be outside the scope for system implementation. If a terminator represents an external device, then there may be a device handler for any input/output between the terminator and the system. Such a handler would constitute a new class or object within the OO design.
- **Data stores.** As with the analysis mapping, data stores should transition to classes or objects, and since they generally represent passive objects within the system solution, additional classes may need to be created to protect the data store, such as a transaction manager or monitor, to ensure mutual exclusion of access to the data. Additional buffer classes may be needed to queue and dequeue data. Nielson and Shumate [NIE88] discuss these types of elements and provide guidance on using OO design and Ada. However, as noted previously, their work is based upon examining the concurrent aspects of design to support real-time and concurrent programming, and not solely to achieve OO designs.
- **Data flows.** Data flows may map to classes or objects, attributes, or specific values.

- **Control flows.** Control flows can indicate some behavioral aspect of the system and are less likely to have representations as classes or objects, attributes, or values.
- **Processes.** If the process is relatively simple and appears to be associated with one specific data flow, then it may map to a class or object operation. Sometimes processes represent complex activities that contain multiple processes, data flows, and data stores. A great deal of care must be exercised when assigning operations based upon DFD processes. At the detailed design level, these operations should map to procedures and functions within the Ada package.

#### 2.4.2.2 Transition Example: Structured Analysis to OO Design

Since the goal here is to produce an OO design of the system, the notation can be an object model from the OO analysis notations discussed previously, which represent classes, or the notation used is the Buhr diagrams to represent Ada structure charts. In this case, as depicted in Figure 15, processes are organized around the data store, Parts Inventory. Delete Part is a component of the process, Send Aircraft Parts, and Add Part is a component of Process New Shipment.

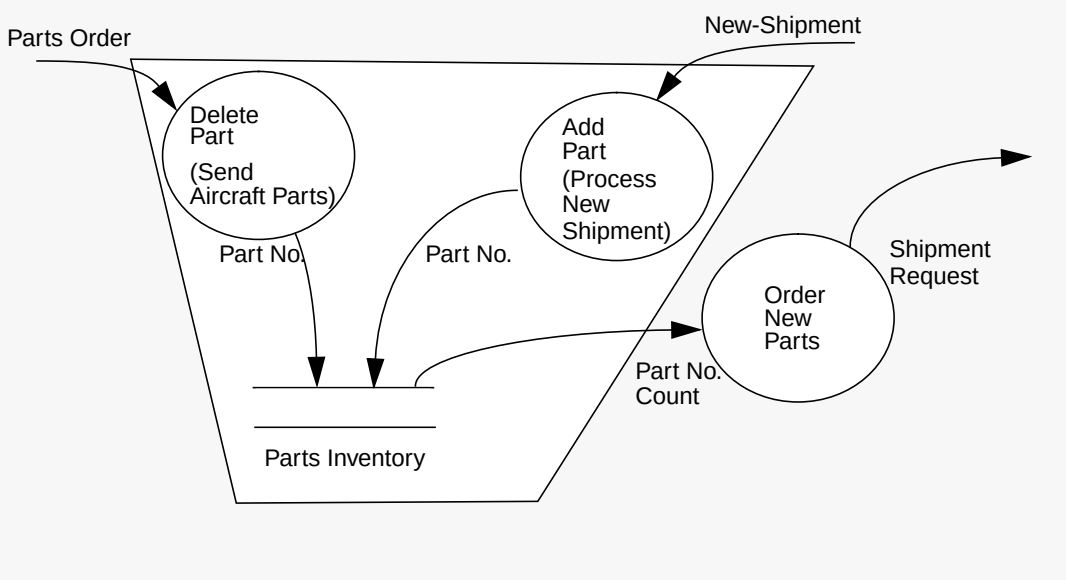



**Figure 15. AMS Data Flow Diagram (Structured Analysis)**

In looking at the design, the AMS can be implemented as a package that provides the interfaces of Process_New_Shipment, Send_Aircraft_Parts, and Order_New_Parts. Packages (Figure 16) are then built around the data store (Parts Inventory). Part and Parts Inventory rep-

resent the same basic object or class of Part. The Part object or class is implemented with two packages, Parts_Manager_Pkg and Inventory_Pkg. Parts_Manager_Pkg acts as a monitor to serialize access to the actual file of Parts, contained in the Inventory_Pkg. The Part class also has the associated operations of Add-Part and Delete-Part, which are implemented as subprograms in Parts_Manager_Pkg and Inventory_Pkg. Other classes, Supplier and Organization may be implemented external to AMS.

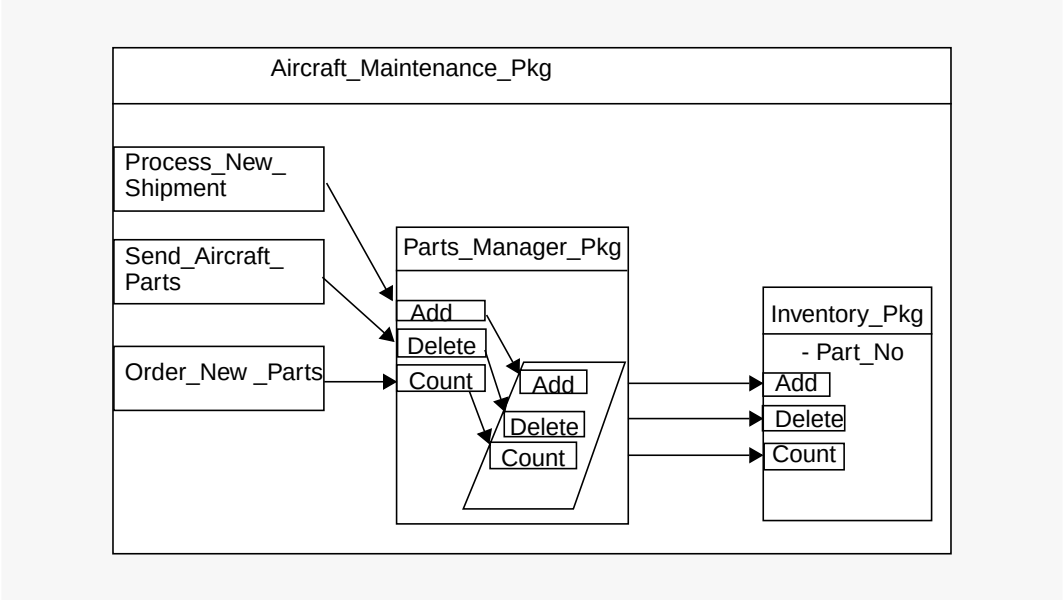



**Figure 16. AMS in OO Design**

# CHAPTER 3. SOFTWARE REENGINEERING EXAMPLE

The Base-Level System Modernization (BLSM) program is an example of OOT use in reengineering a DoD information system. Its mission is to modernize base-level information systems at U.S. Air Force bases. The low level of interoperability and difficulties in maintaining the current, standard base-level information systems are principal drivers of the BLSM program. Most of the information systems are between 20 and 30 years old. The software has become more difficult and expensive to maintain because of the magnitude of the code changes made over time. Originally, specific functional requirements drove the designs of these information systems and most still work fairly well. However, many of the information systems no longer meet the needs of the user because of mission changes and advances in technology, and modification has become increasingly difficult. As a result, these legacy systems need to be reengineered to take advantage of modern software engineering principles and to meet current user needs.

The age and design of the software, databases, and interfaces, the dependency on proprietary operating systems and hardware, expiring hardware contracts, the availability of new technology to assist in development and maintenance, and the need for interoperability are all documented in the requirements recorded in the Statement of Operational Need (SON) [USAF89] and the HQ USAF/SC Program Management Directive (PMD) [USAF92] for System Modernization for Standard Base-Level Communications-Computer Systems (also captured in [BLSM93]).

Most of this chapter is extracted directly from BLSM project reports.

## 3.1 BLSM MISSION

The mission of BLSM is to support the process reengineering of base-level systems, providing a consistent architecture designed to survive 20 to 30 years and ensuring complete interoperability among the information systems. A *standard system* is a system used by more than one Major Command (MAJCOM) and is supported by centralized management. The domains covered by this program are in the functional areas of base civil engineering, services, comptroller, logistics plans, supply, maintenance, medical, operations, manpower, contracting,

personnel, and transportation. Additionally, the following functional areas may be involved in BLSM: communication-electronic, administration, data automation, materiel management, security police, and information management [BLSM93, p. 4].

## 3.2 PROGRAM OBJECTIVES

The overall objectives of BLSM are to create standard information systems which can be used effectively during war and in peacetime, to provide a human-computer interface which minimizes the cost and duration of user training, and to move all the standard base-level systems into an OO and open systems environment.

More specific program objectives include the following [BLSM93, pp. 7-9]:

- **Ensure user satisfaction.** The DoD Technical Architecture Framework for Information Management (TAFIM) and the BLSM PMD both call for increased user productivity through the use of standard, consistent user interfaces and integrated applications that share data. The BLSM program is developing systems with consistent interfaces that share data, fulfill the user's current needs, and are flexible enough to meet expanding requirements.
- **Improve business practices.** To meet this objective, the BLSM program is managing information through centralized control and decentralized execution, validating new methods prior to implementation, simplifying by elimination and integration, and continually re-examining and redefining to improve operations.
- **Decrease life cycle costs.** The PMD calls for a decrease in life cycle costs by eliminating and reducing maintenance and training costs.
- **Develop and use reusable components.** The BLSM methodology provides high potential for life cycle reuse by providing a separate group whose responsibilities include reviewing all object models during initial phases and creating abstractions that are understandable and reusable by many systems. BLSM will also use the products from established DoD programs to meet its needs. Software modules and components are stored in the Defense Software Repository System (DSRS) for future use.
- **Develop a single logical database.** Data standardization and the methodology to perform database design in an OO methodology are fundamental in the effort to create a single logical database. An enterprise analysis model documents the data integration of common objects, classes, and data elements at a logical level.

- **Implement open systems standards.** BLSM is establishing the process and methods for implementing open system, OOT, and reuse.
- **Develop scalable and flexible systems.** BLSM projects will be developed for flexibility to changes in the military environment (i.e., doctrine, force structure, political changes, base closures). By conforming to and supporting various modes of operation, BLSM will support one to many users and one to many wings—in other words, *scalability.* Establish a software engineering environment. BLSM is using the best available software engineering tools and techniques combined with management policies, decision points, and activities to establish a state-of-the-art software engineering environment.

## 3.3 AIR FORCE OPERATIONS RESOURCE MANAGEMENT SYSTEM

BLSM is the umbrella program for the incremental modernization of all Air Force standard base-level systems. Three lead BLSM application systems were elected to develop and refine the BLSM methodology, establish the technical support base (e.g., skill base, computer-aided software engineering (CASE) tools), and implement the technical-management infrastructure (e.g., risk management and reuse programs) prior to full-scale BLSM implementation [HARR93, p. 1-1]. We have selected one of these applications, the Air Force Operations Resource Management System (AFORMS), to serve as a model example of BLSM's approach to full-scale software reengineering using OOT.

AFORMS is an existing system that provides operations management information to operations supervisors to support effectively the implementation of Air Force flight management policies. This system ensures that the status of Air Force flying programs is available to assist operations managers in making resource allocation decisions. AFORMS ensures accurate tracking of flying and ground training programs for each weapons system at each base; availability of flying program statuses to allow immediate analysis and effective resource allocation decisions; and effective base-level, MAJCOM, and Air Force support for weapons systems requirements [HARR93, p. 1-2]. Figure 17 provides a high-level overview of AFORMS architecture.

The presentation of the BLSM approach here will both outline the general BLSM methodology and illustrate the application of that methodology to AFORMS, with a focus on the OO aspects of this reengineering program. It begins with discussion of an enterprise analysis for the entire BLSM program, proceeds to provide an overview of the general BLSM software engineering environment, and then focuses on structure of individual software development

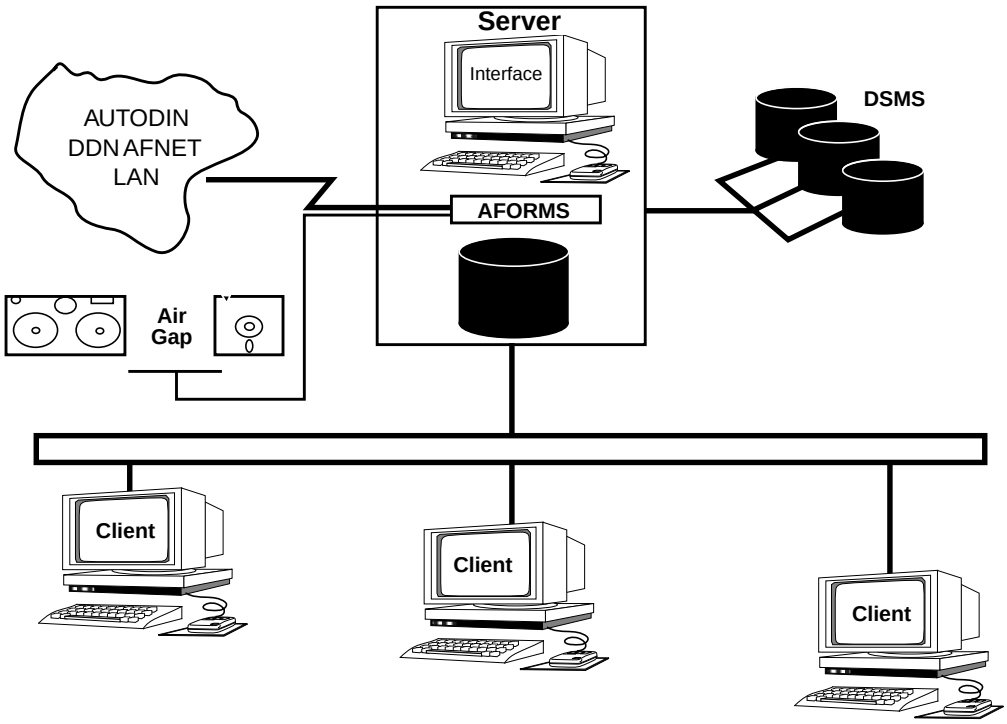



**Figure 17. AFORMS Architecture Overview**

phases for specific applications. These phases are illustrated with examples of OOT usage from AFORMS.

## 3.4 ENTERPRISE ANALYSIS

BLSM is a broad program, covering 12 functional areas with 36 major automated information systems affecting all active USAF bases [BLSM93, p. 4]. Its effective pursuit requires a broad analysis of the existing enterprise to identify commonalities among different areas, establish standard object models for them, and to set priorities for modernization. The entire base-level computing environment is the enterprise analyzed under the BLSM umbrella. Several external areas also serve as input to the BLSM enterprise analysis in order to ensure cooperation and harmony with outside efforts and requirements [BLSM93, p. 42].

### 3.4.1 Sources of Analysis Input

The external inputs to enterprise analysis include data modeling efforts from other areas at the Standards Systems Center (SSC), the United States Transportation Command (USTRANSCOM), the Theater Battle Management Group (TBM), the Defense Information Systems Agency (DISA), and the DoD functional communities. BLSM uses the output of a

Wing-Level IDEF0 and IDEF1X models to bring the Air Force enterprise together, identify the current state of the Wing business process, and to identify areas for potential business process improvements. Other requirements which may not be identified by the Wing-Level IDEF models but are critical to the enterprise analysis are MAJCOM, Air Force, and DoD directives, policies, and guidance, as well as Congressional policies and laws which must be supported by the Standard Base-Level Computing Systems [BLSM93, p. 44]. Some of the policies used as input include Defense Management Review Documents (DMRD), the DoD TAFIM, the DoD Technical Reference Model (TRM), the Open Systems Environment for Imminent Acquisitions (OSE/IA), and the policy on use of Ada [BLSM93, p. 43].

### 3.4.2 IDEF Business Process Models

The business process changes identified by the Wing-Level IDEF models are a critical input to modernization of the BLSM enterprise. The restructuring of the Air Force into Composite and Objective Wings and the long-term goals of the program dictate the development of a "system of systems" that supports war fighting needs, as well as the day-to-day operations of the Wing. The output of the Wing-Level IDEF models is a critical input to modernization of the BLSM enterprise, as well as any needed restructuring of existing processing capabilities. The outcome of this process identifies the priorities for the next step, the definition of functional domain IDEF models.

IDEF models of each of the functional domains are generated to ensure that modernized information systems support newly identified requirements and are adaptable to future computing needs. The computing needs of the functional community are optimized and incrementally developed, based on the functional domain model [BLSM93, p. 43].

### 3.4.3 Software-Level Analysis

A software-level analysis of the existing data and processing requirements supported by the Standard Systems Center at Gunter Air Force Base, AL, provides the information necessary to develop an object model and pieces needed to develop interoperable systems. The objective of this analysis is to create a single information model which supports the Wing-level and functional area business process analysis and improvement efforts. This analysis identifies and models many of the objects, components, and modules necessary to actively support minimizing and eliminating duplicated processing and data across the wing as defined by the IDEF process.

However, analyzing the information requirements of all information systems at one time is a monumental undertaking. Resolution of this problem involved selecting a key information system from each functional area as a starting point for the analysis. The remainder of the information systems are analyzed as they are modernized, according to the BLSM development schedule [BLSM93, pp. 44-45].

### 3.4.4 OO Analysis

A comparison of the existing data elements and processing in each of the information systems is used to identify the common needs of the functional domains. Base objects in the object model are generated by grouping data elements and processes based upon frequency of use and common associations with domain objects (e.g., aircraft, organizations, equipment, events, accounts, etc.). An OO analysis is accomplished once all information is grouped into objects with relationships between objects, such as inheritance.

Once the object requirements are identified, libraries are built using a prioritization of the objects based on statistics and the BLSM development schedule. During the analysis process, statistics on which elements are used by the most information systems were gathered. Table 2 lists some of the objects that were found to have an extremely high level of commonality across BLSM information systems. The gathered statistics prioritize which objects are developed first. If an object has a high incidence of commonality across the information systems, it receives a high priority for development. The BLSM development schedule also plays a large role in determining priorities to ensure the availability of objects required by the first information systems going through the development cycle [BLSM93, pp. 45-46].

### 3.4.5 Products of Analysis

The products from the enterprise analysis are as follows:

- The object model based on the software-level analysis of enterprise-wide existing data and processing requirements;
- A schedule for prioritized object development;
- Identification of common use components and modules across the BLSM enterprise;
- A schedule for prioritized component and module development; and

- An incremental development schedule of information systems based on available pieces and the needs of the user community as identified by the Wing-Level and functional area IDEF models [BLSM93, p. 44].

**Table 2. List of Potential High-Commonality BLSM Objects**

| | | | |
|---|---|---|---|
| accounts | dates | installation | places |
| addresses | engines | inventories | receipts |
| aircraft | equipment | invoices | reports |
| allocations/authorizations | errors | items | requests |
| assignments | evaluations | itineraries | schedules |
| cargo | events | jobs | shipments |
| catalogs of information | examinations | maintenance tracking | shipping agent |
| classes/courses | files/records | manifests | simulators |
| clearances/security | flight information | mission data | student information |
| configurations/profiles | fuel | mobility | task and work breakouts |
| containers | funds | owners | time and time zones |
| contracts | government bills of lading | passengers | training |
| conveyances | guns/weapons | pay information | transactions |
| countries | hazards/special handling | payments | units/organizations |
| crews | helicopters | personnel information | vehicles |
| customers | inspections | phone numbers | vendor information |

## 3.5 SOFTWARE ENGINEERING ENVIRONMENT

Figure 18 shows the computers used in the development environment, the activities performed on each computer, and the software tools that execute on each computer. The software engineering development environment consists of three primary platforms:

- Unix-based workstations used for analysis activities, documentation, window code generation, design activities, and prototypes;
- Rational computers used for the design, coding, testing, and configuration management of Ada software; and
- VAX computers used to host the relational database and perform configuration management.

In addition, personal computers (PCs) are used to support planning, storyboarding, and graphics activities. All computers are connected via a local area network (LAN) using TCP/IP (Transmission Control Protocol/Internet Protocol). The target computer environment is also connected to this LAN. The number and capacity of each platform in the development environ-

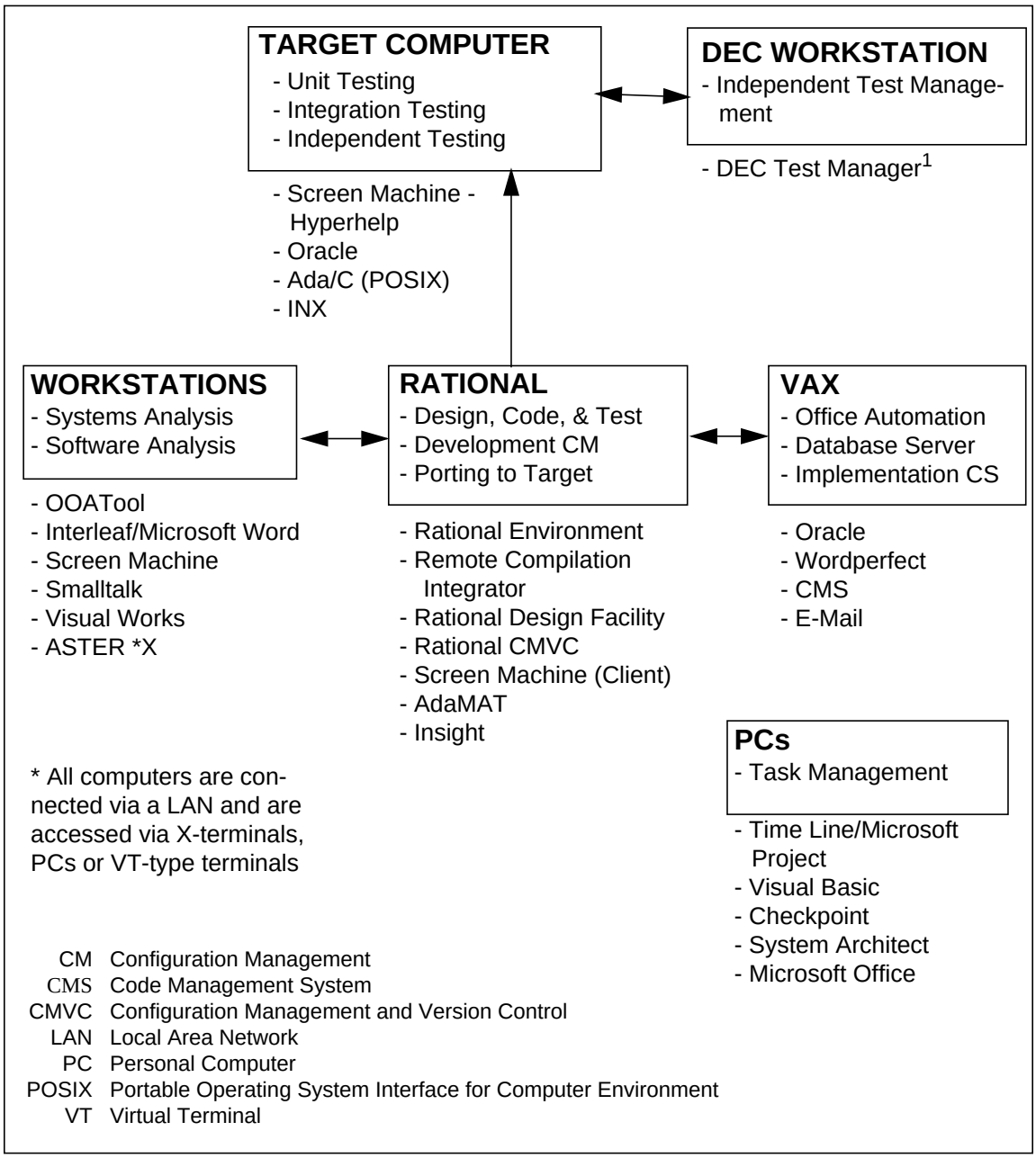



**Figure 18. Software Engineering Development Environment**

ment is dependent on the number of developers and the size of the application being developed. For example, a VAX 4000 is used for the Logistics Module - Base Level (LOGMOD-B) project, while a VAX 6000 is shared between AFORMS and the Manpower Data System (MDS) projects.

At the beginning of the software development life cycle, the analysis team and Systems Engineering Group use workstations to perform system and software analysis using OOATool

and Smalltalk. Each tool takes advantage of the strong graphical capabilities and computing power of the Sun workstation. Screen Machine is used to develop windows during the Software Analysis Phase.

As the project moves through design, the analysis team electronically transfers information from the OOATool model to the Rational computer. This allows for a direct relationship between the Ada design information on the Rational and the analysis information developed on the workstation using OOATool.

The Rational platform is used for the designing, coding, unit testing, and configuration management of the Ada application. The Rational environment and support tools (e.g., Design Facility) are used by the software engineers to automatically generate the requirements and design documents (i.e., Software Requirements Specification, Software Design Document) from information maintained in the OOATool. Coding and testing of increments are performed by the task team using the Rational editors and debugging tools. The team also uses Screen Machine on a workstation for the human-computer interface (HCI) and Oracle on the VAX as the relational database management system (RDBMS). During execution, remote procedure call (RPC) mechanisms allow the application on the Rational to directly access Oracle and Screen Machine on the workstation.

Once the increments of the application are coded and tested, software can be ported and tested in the target environment. This early porting lowers the risk of encountering portability problems during final integration on the target computers. The Independent Systems Test Group performs testing on the target computer while using the DEC Test Manager for regression testing.

**Software Items**

A description of each software item used in the software development process follows. Figure 19 illustrates which tools are used during which software development phase.

a. **OOATool.** Used to model requirements and the design of the system. It supports the OO methodologies used during BLSM. Class and object models developed in O0ATool are used in the development of the software requirements and preliminary software design.

b. **Rationale Environment.** The primary development system used in designing, coding, testing, and integration of BLSM systems. This environment provides an array of CASE capabilities such as configuration management; documentation generation

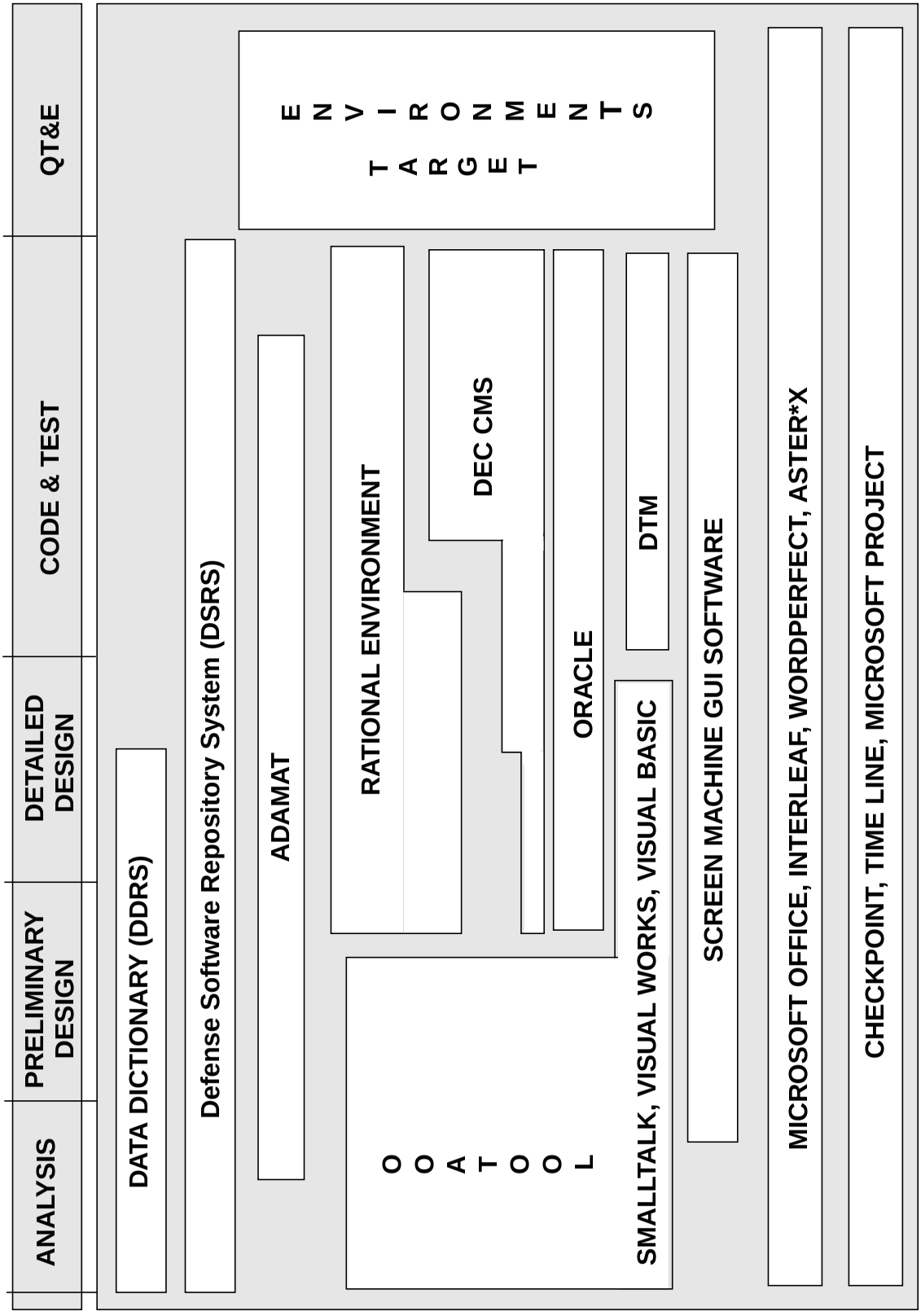



**Figure 19. BLSM Software Development Tools**

and requirements tracking; an automated, incremental target build facility; Ada library incremental compilation, synchronization, and status; semantic editing; and automated enforcement of coding standards.

1. **Rational Configuration Management and Version Control (CMVC).** Rational's sophisticated configuration management tool set. Once objects (text files or Ada units) are placed under CMVC control, their change history from that point on is recorded. Change history includes the tracking of software changes during development and allows regression when needed. Any object may have historical information tracked. CMVC also provides a mechanism for making a "release" of a Rational subsystem. This freezes a snapshot of the subsystem in time.

2. **Rational Design Facility (RDF).** A layered product which supports the development of DOD-STD-2167A documentation. RDF provides capabilities such as document generation and requirements tracking.

3. **Remote Compilation Integrator (RCI).** Supports universal host software development. This utility can be used to support transfer of software to the target computer and the generation of target machine compilation scripts.

c. **VAXset.** A collection of DEC tools that support the software development process and all of the DEC-provided programming languages. This tool set includes the Code Management System (CMS) and DEC Test Manager (DTM).

1. **CMS.** Provides an efficient method for storing files in a project and tracking all changes to those files—it keeps track of files at every stage of development. CMS provides for mutual exclusion, generates project activity reports, maintains a history of library activity, and can store files created by other DEC tools. CMS maintains the electronic Software Development Files (SDFs).

2. **DTM.** Provides flexibility in organizing tests, in selecting tests for execution, and in reviewing and verifying test results. DTM also automates the regression testing process by comparing established benchmarks with tests.

d. **VAX C Compiler.** Provides support for developing C programs that will be used for particular Ada bindings (e.g., Ada/SQL binding). It is fully integrated with the VAX debugger and VAXset tools.

e. **Oracle RDBMS.** Provides an relational database management system for use in the development environment. A portable Ada/SQL binding will be used to allow application software to communicate with Oracle.

f. **Screen Machine.** Provides tools for building user interfaces that include pull-down and pop-up menus, action buttons, and complex data entry forms. Screen Machine eliminates the need for an Ada binding to a C-based graphical user interface (GUI).

g. **AdaMAT.** A static code analyzer designed for use with the Ada language. The user can assess the overall quality of software. AdaMAT monitors more than 150 parameters and produces reports which identify quality problems in the code and isolates their causes. AdaMAT also helps determine the difficulties of porting and maintaining the code. It is a general tool that can be used by different functions in different ways. For example, the test manager can determine the minimum quality levels for the code to meet prior to acceptance for testing.

h. **Checkpoint.** Resides on a DOS-based machine and provides three capabilities: l) project assessment, 2) measurement and analysis, and 3) estimating. The Checkpoint user can make assessments of project status at various points in the projects life cycle against industry standards using function points, feature points, or lines of code. It also supports measurement and analysis, including the analysis of quality, productivity, and defect removal efficiency. Another application, System Evaluation and Estimation of Resources (SEER), has been adopted by the Air Force and is currently under evaluation to replace Checkpoint.

i. **WordPerfect.** Provides word processing capability to anyone on the network. WordPrefect is used for the generation of textual material to be used in documents and briefings. Working files of required deliverables are drafted in WordPerfect and then exported to Interleaf for finalization and delivery.

j. **Interleaf.** A document production/publishing system that supports integrated text and graphics. Interleaf is used for the publication of program documentation during all phases of development.

k. **Microsoft Project.** A project-planning tool on the DOS-based machines used to maintain the top-level and detailed schedules for the BLSM effort.

l. **ASTER*X.** An office automation tool which provides word processing, spreadsheet, and graphics functions on the Sun workstations.

m. **Smalltalk.** Used as a prototyping tool. It supports inheritance and polymorphism to enable the design of fully OO modules and comes with a complete class library for quick prototyping. Visual Works is a front end to the Smalltalk prototype which produces the human interface.

n. **Visual Basic.** A tool used to "storyboard" screens. Visual Basic is used on the DOS-based machines.

o. **Insight.** A design tool on the Rational which produces Booch diagrams.

p. **System Architect.** Used to produce the data modeling diagrams and to automatically generate the schemas from the models.

Sections 3.7 through 3.16 describe at a high level what activities take place during each phase of software development. Each description is accompanied by a diagram showing all components of that software development phase. It is important to note that throughout the software development process the DSRS is continually queried for, and updated with, reusable components. It should also be noted that, although the following phases are discussed end-to-end in a sequential manner, they are used to develop systems incrementally by looping back through certain phases. Figure 20 depicts the incremental development loop within the software development process.

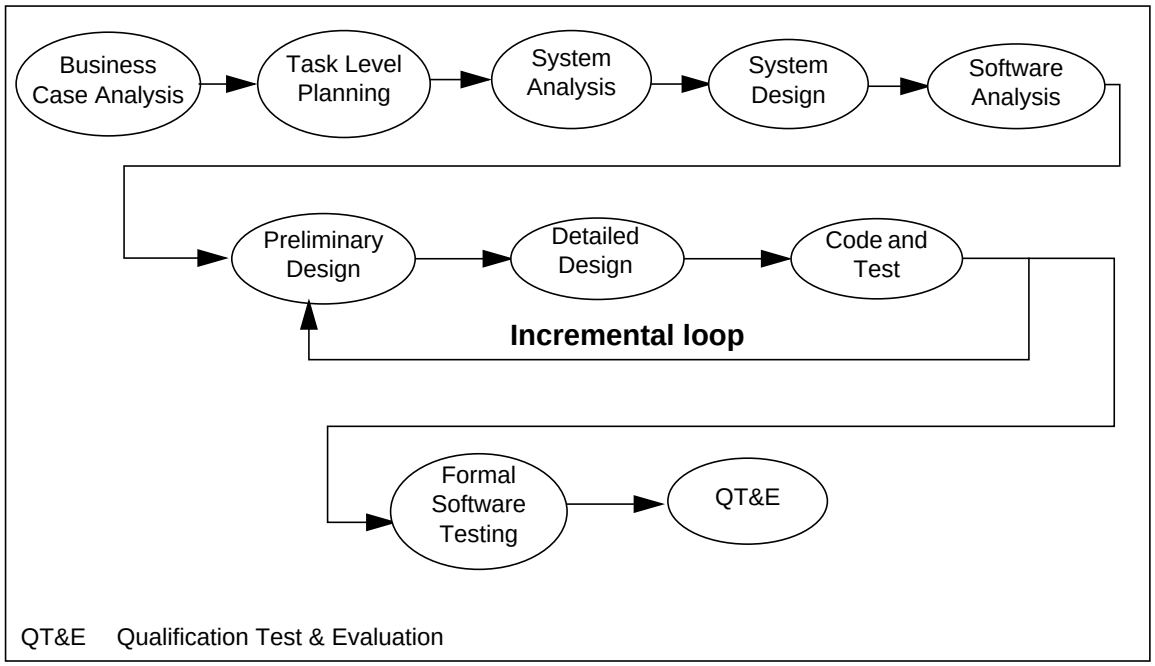



**Figure 20. BLSM Software Development Process**

## 3.6 BUSINESS CASE ANALYSIS PHASE

The purpose of this phase is to review each of the processes in the functional program to determine what the current "as is" processes are, evaluate the validity of each process and then improve it prior to determining what will be automated

This process contains a myriad of subactivities under each of the major activities listed in Figure 21. The process modeling is supported by IDEF0 and IDEF1X models as well as cost-

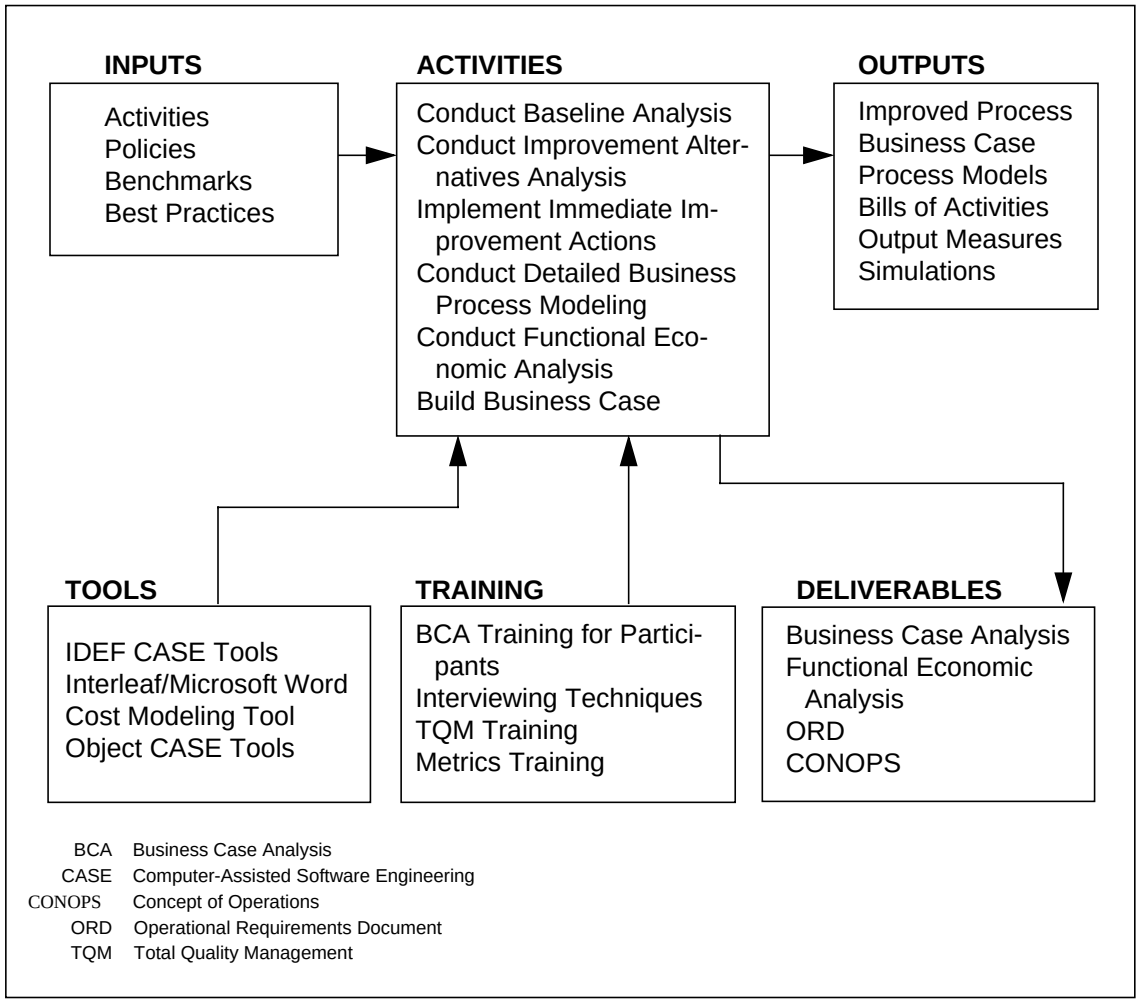



**Figure 21. Business Case Analysis**

ing tools such as spreadsheets and economic models. The following subactivities occur under each of the major activities in accordance with the Corporate Information Management (CIM) Business Process Improvement (BPI) process.

## 3.7 TASK-LEVEL PLANNING

Task-Level Planning includes all activities leading up to the System Analysis Phase which was previously depicted in Figure 20 on page 43. The major activities are illustrated in Figure 22.

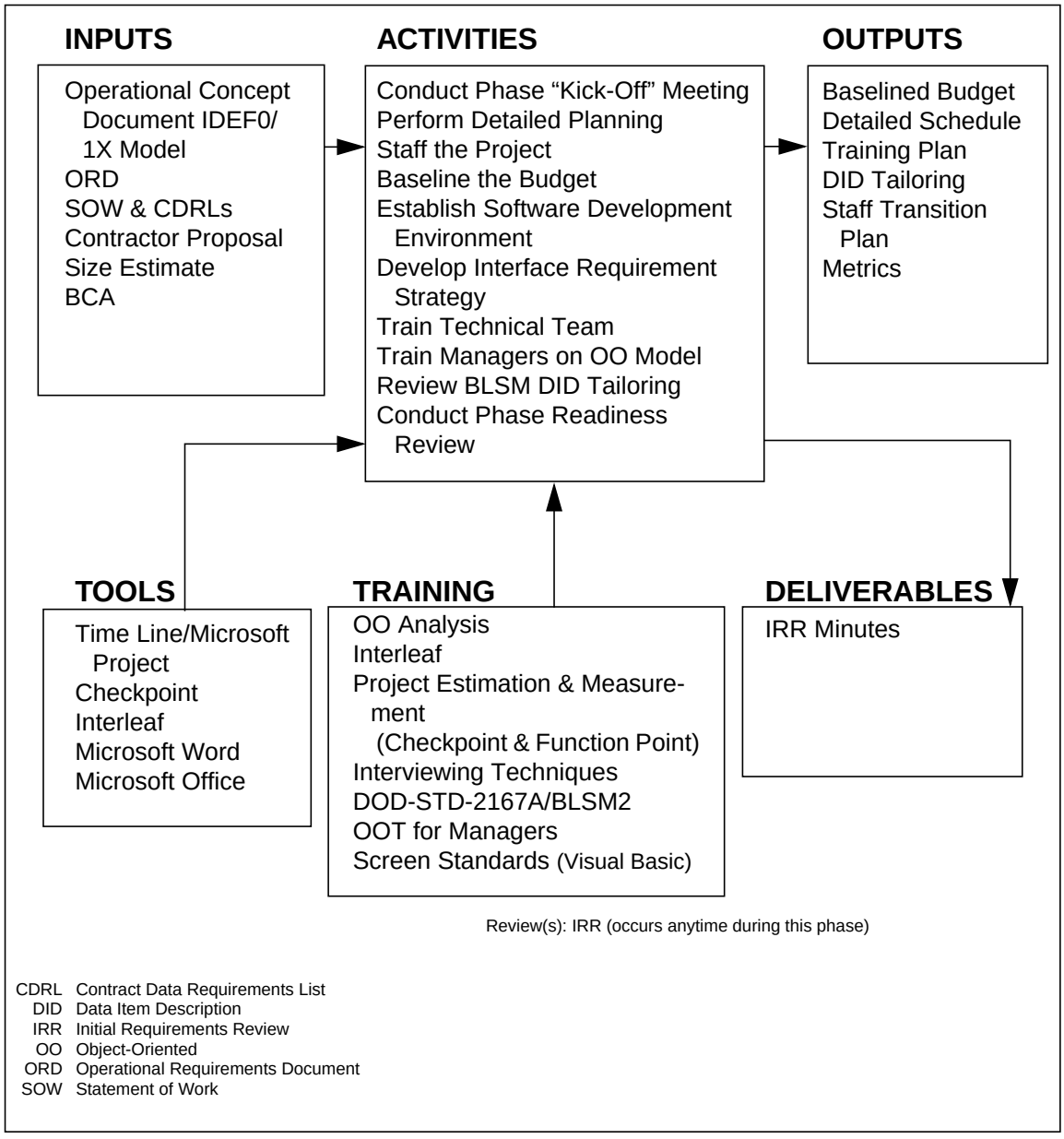



**Figure 22. Task-Level Planning**

One of the major outputs of this phase is a detailed schedule showing major milestones, development phases, delivery dates from the CDRL, and training course dates. A detailed plan

for staffing the task is also produced, indicating when staffing increases and when individuals will become available to other tasks.

The Initial Requirements Review (IRR) is held during the Task-Level Planning phase to brief the Government on the proposed scheduled activities and methods for modernizing the system.

OOT use is represented primarily by training and by tailoring of the DIDs in preparation for subsequent phases in which DIDs are generated based upon the objects and classes developed in the OO analysis model.

## 3.8 SYSTEM ANALYSIS

The System Analysis phase determines the system-level requirements of the problem domain (or functional area) to be modernized. The major activities of this phase are illustrated in Figure 23.

The Visual Basic tool is used during this phase to demonstrate or storyboard the system windows to the users to gain their involvement and confidence early in the development process. Smalltalk training with an emphasis on prototyping, in particular the patterns needed to implement a construct from the OO analysis model, will be provided during this phase.

At the conclusion of this phase, the integrated team holds a System Requirements Review (SRR) to determine the acceptability of (1) the system requirements as documented in the draft System/Segment Specification (SSS) and (2) the system design as documented in the draft System/Segment Design Document (SSDD).

One of the main outputs of this phase is the system-level OO analysis model of the problem domain. At this point in the task's development cycle, the OO analysis model contains preliminary subject areas of the problem domain and major functional requirements of the system.

OOT comes into focus in this phase as work is begun on the OO analysis model. In AFORMS, this required dividing the model into subject areas that are aligned with the four major functional areas of the system (*Resource*, *Training*, *Hours*, and *Schedule*) and three support areas (*System Utility, External Interface*, and *Report*). Figure 24 depicts the relationship between these seven subject areas and each area’s sets of class and object(s).

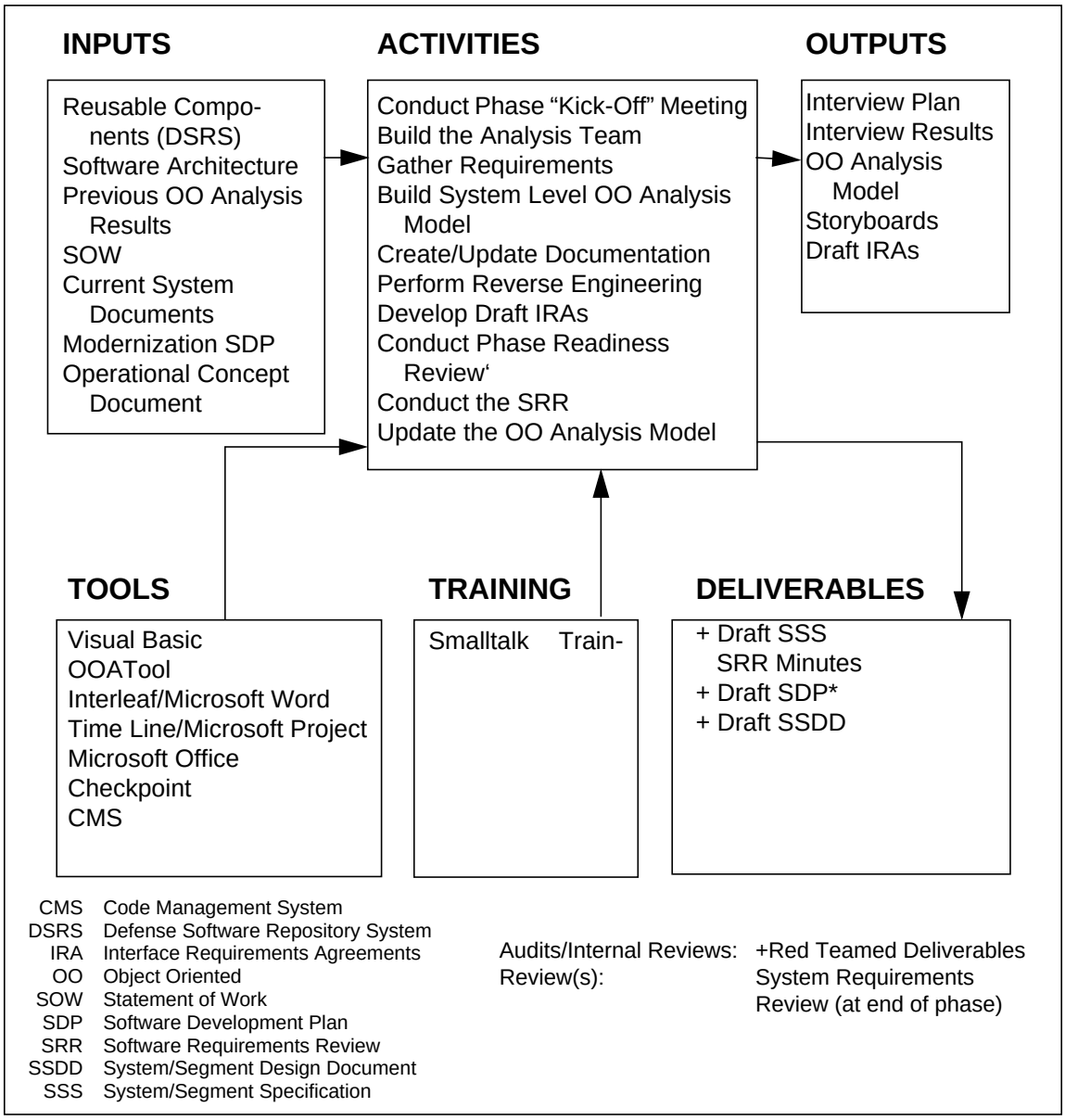



**Figure 23. System Analysis**

## AFORMS Subject Areas and Classes

The following description of AFORMS is by subject area grouping and gives a brief overview of the function of the system and those objects that constitute each function. The complete list of objects for each subject area has been previously depicted in Figure 24 on page 48.

*Resource Subject Area.* The Resource subject area represents the core of AFORMS by providing the capabilities required to manage human flight resources. It allows for the admin-

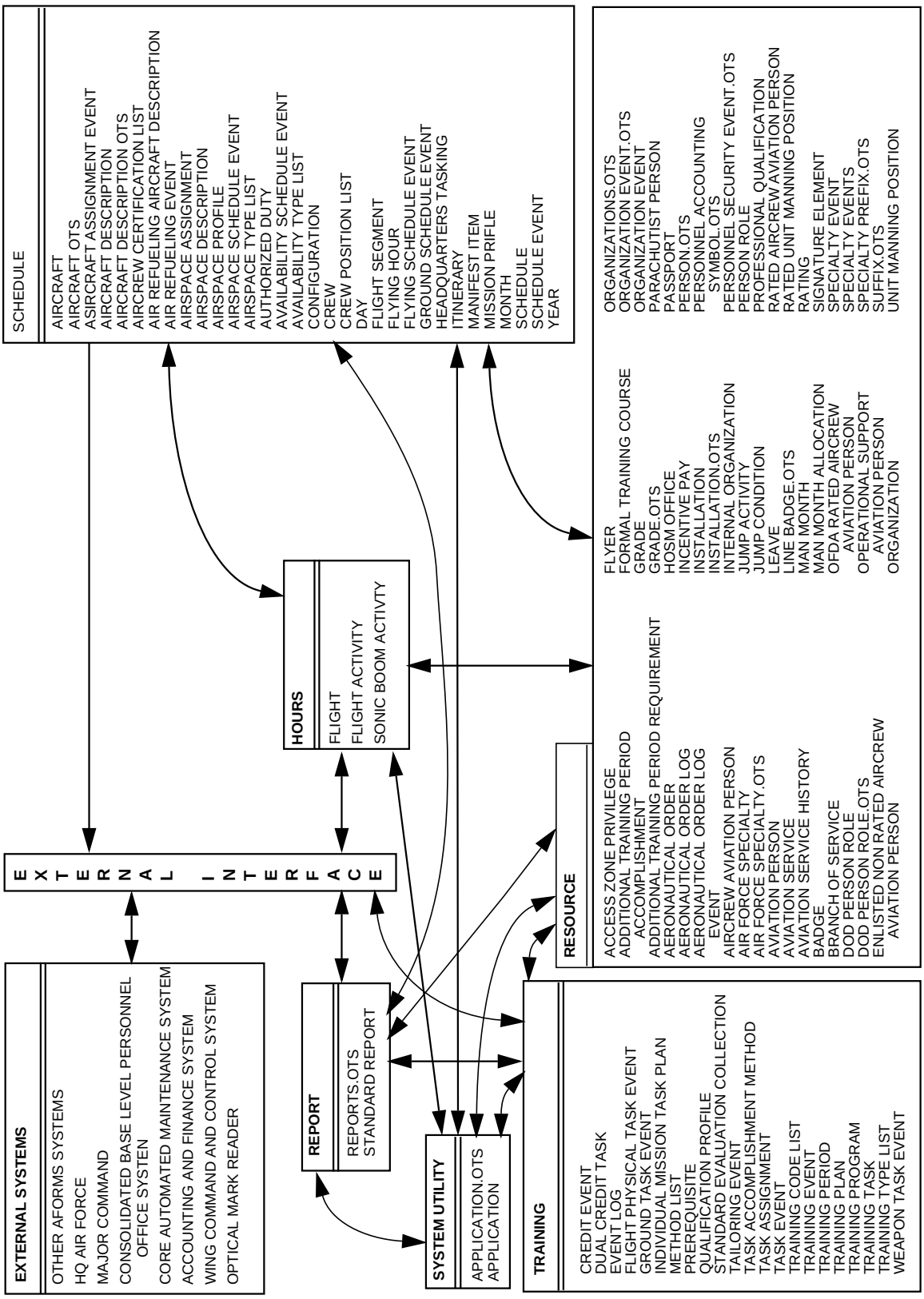



**Figure 24. AFORMS CSCI Overview**

istration of USAF flight management policies while supporting the Aviation Career Incentive Act (ACIA) by maintaining those records required to qualify personnel for Aviation Career Incentive Pay (ACIP) and Hazardous Duty Incentive Pay (HDIP).

The Resource group, coupled with Hours and Schedule, allows AFORMS to monitor individual flight pay entitlement and initiates Military Pay Order (MPO) actions to start and stop flight pay. Additionally, AFORMS automates the preparation of aeronautical orders for changes to entailment status, aeronautical ratings and for the award of badges. The result is the capability to monitor an individual's aviation career.

*Training Subject Area*. The Training subject area provides the capabilities to specify training requirements and build the training plans required for each air crew member. These training plans can be individualized and used to track accomplishments. The relationship between the objects of this group and those of Resource, Schedule, and Hours provides AFORMS with the capability to maintain continuation and additional training plans for individuals, thus eliminating redundant record keeping caused by the difference between the Air Force standard and MAJCOM unique systems.

*Hours Subject Area*. The Hours subject area provides the capabilities to maintain information on flights, flight evaluations, and sonic booms. Through connections between objects within this area, and with connections to objects within the Resource area, the AFORMS CSCI (computer software configuration item) will create and maintain aircraft flight activity records, career total records, and an active flight record for each human resource.

*Schedule Subject Area*. The Schedule subject area, in conjunction with the Training and Resource areas, provides the capabilities to build flight training schedules and to monitor an individual’s adherence to these schedules. In addition, this area will produce products that will assist schedulers in assigning air crews to a mission and in checking for scheduling conflicts. Information concerning scheduled activities will be stored.

*System Utility Subject Area*. The System Utility support area provides general system capabilities such as installation and shutdown procedures and *ad hoc* report generation. These capabilities will be met by reusable software components to be identified and obtained via the Common Utilities/Bindings Task Order.

*External Interface Subject Area*. The External Interface support area provides the external interface capabilities for AFORMS. The OO analysis model shows those systems outside of AFORMS interfaced with as objects that are associated with the External Systems. The External Interface subject area consists of a single class and object.

*Report Subject Area*. The Report support area provides the capabilities to produce standard reports. These capabilities will be met by reusable software components to be identified and attained via the Common Utilities/Bindings Task Order.

## 3.9 SYSTEM DESIGN

An important activity of the System Design phase, as depicted in Figure 25, is identifying what system requirements can be fulfilled by bindings, commercial off-the-shelf (COTS) software, hardware, and manual procedures. To aid in this analysis, reuse repositories are queried to determine if any existing software architectures, bindings, previous OO analysis results, or Corporate Object Model objects can be used to satisfy any of the system requirements.

The OO analysis model is further refined during System Design. The model's objects depict things or components that the problem domain contains and is responsible for. Examples of objects include events, reports, people, equipment, authorizations, and organizations. Objects are described by their "attributes." Their "services" define the processes they are responsible for performing.

During this phase the SSS and SSDD are finalized and the Software Requirements Specification (SRS) is drafted.

The System Design Phase concludes with the System Design Review (SDR) and the establishment of the Functional Baseline. The initial function point count for the system is reported at SDR.

## 3.10 SOFTWARE ANALYSIS

The Software Analysis phase, depicted in Figure 26, is characterized by acquiring a detailed understanding of problem domain software requirements. When this phase is finished, the OO analysis model will be complete and contain not only the objects, their structures, and services, but the objects attributes (data element names), and all messaging connections between the objects.

An important activity of this phase is using the Smalltalk tool to prototype the model of the problem domain component as it is being built. The use of Smalltalk helps ensure that necessary attributes, services, and messages are defined to permit the finished system to perform its intended functions. The Systems Engineering Group performs periodic audits on the BLSM task's OO analysis models and determines if there are candidate objects for the Corporate Object Model. If the candidate objects are determined to be reus-

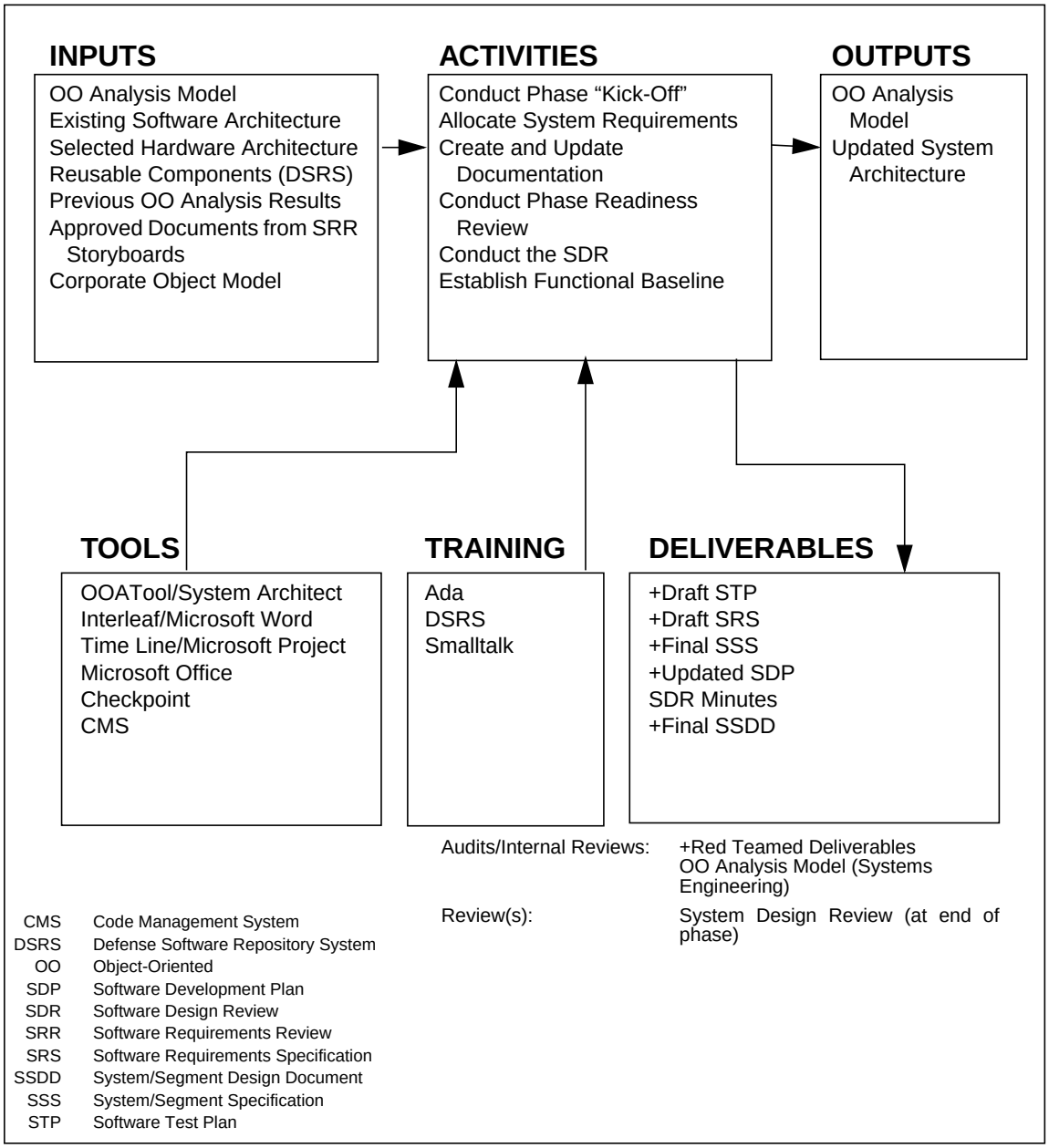



**Figure 25. System Design**

able, they are moved to the Corporate Object Model. Abstracts for all objects are submitted to the DSRS. The reuse repositories are used heavily by the analysis team as they are queried for possible reusable components, previous OO analysis results, and objects. In turn, the repositories are updated throughout the Software Analysis phase with the completed OO analysis model, approved data elements, and new corporate objects.

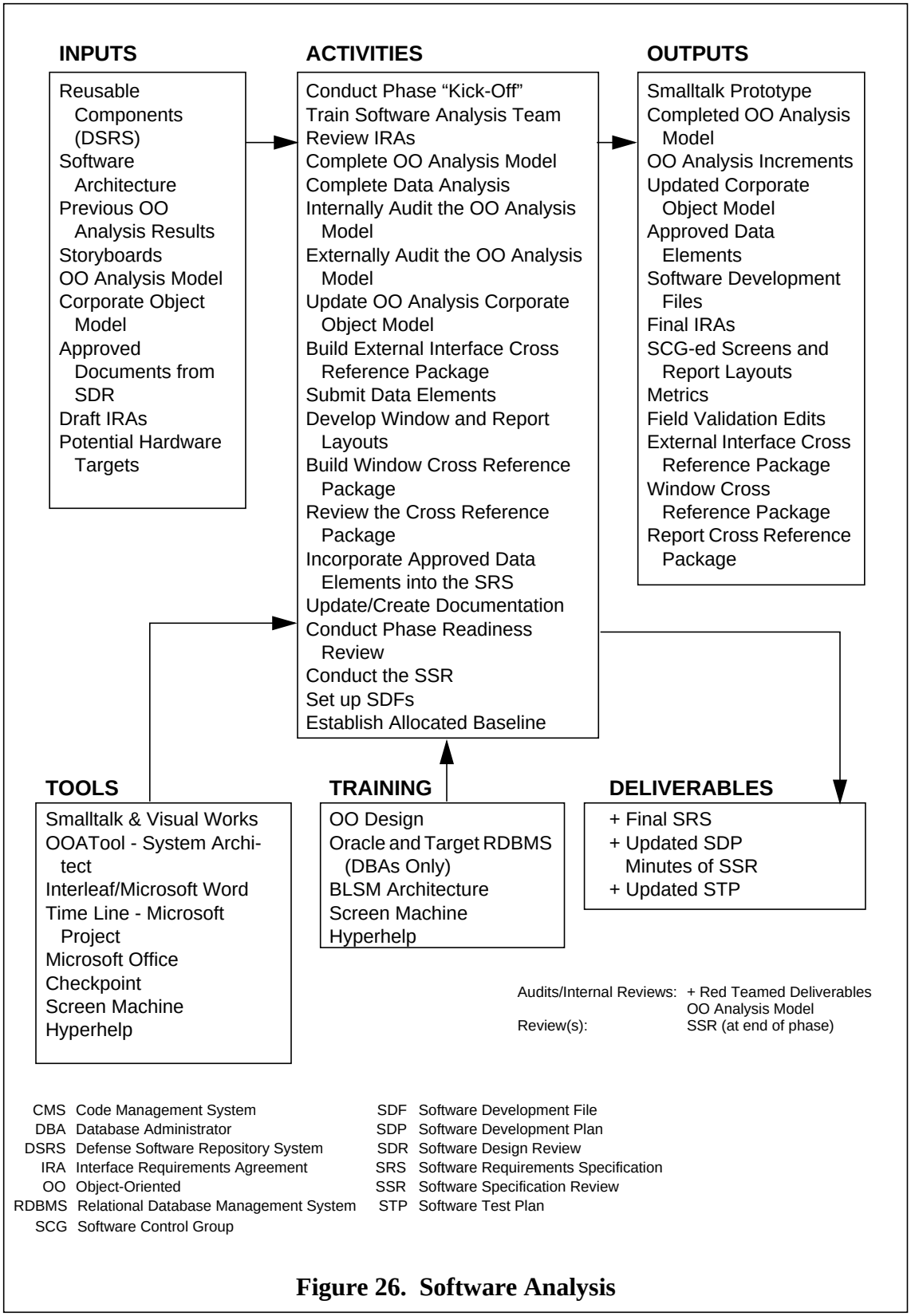



**Figure 26. Software Analysis**

Windows and report layouts are developed by the SSR and baselined with the SCG. Preliminary design begins on risk areas of the Problem Domain Component (PDC) of the OO analysis model. During the time allotted for each BLSM task to review and approve the SRS, the preliminary design of the PDC can continue. In addition, the OO analysis increments must be defined and reported to the task leader for planning purposes.

Software Analysis concludes with the SSR and the establishment of the Allocated Baseline and the creation of the Software Development Files (SDFs). The function point count, which was first reported at SDR, is updated and reported at SSR.

### 3.10.1 OO Analysis Model and Information Management Model

At the completion of Software Analysis phase, the entire system and software requirements are depicted on the OO analysis model. A small portion of the Coad-Yourdon graphical format [CDYD91] of this model is illustrated in Figure 27. Object icons are double boxes with object and/or class name at the top, followed by the attributes of the object and the services (or methods) provided by that object. The full model graphically depicts all problem domain objects, their relationships (structure) to other objects, the services (processes) each object is responsible for performing, the messages necessary to communicate between objects, and the attributes (data elements) which describe each object. Any approved data elements (attributes) for the system will reside in the DDRS and will be delineated as an attachment to the SRS.

At the same time, and from the same analysis as the OO analysis model, the Data Administration member of the Software Analysis Team develops the Information Model (see Figure 28). Transition from the OO analysis model to the Information Model requires distinguishing between data objects and user objects in the OO analysis model. Data objects are the objects stored in the database. User objects are the objects derived by the application that do not require persistence. The Information Model consists only of data objects. The Information Model, that is, data objects, also differ in their exclusion of the services and/or methods that are essential parts of some OO analysis model objects. Some objects in the OO analysis model can be consolidated into a single data object. Objects can be combined if and only if the single data object can portray all of the functionality of each of the separate objects.

After the Information Model is created, determine what attribute or attributes make each of the data objects unique. This is a candidate logical identifier for the data object. In some cases, a data object will have multiple candidate logical identifiers. In other cases, the data object will not have any candidate logical identifier. If a candidate logical identifier cannot be determined, identify which attributes are used to retrieve the stored object.

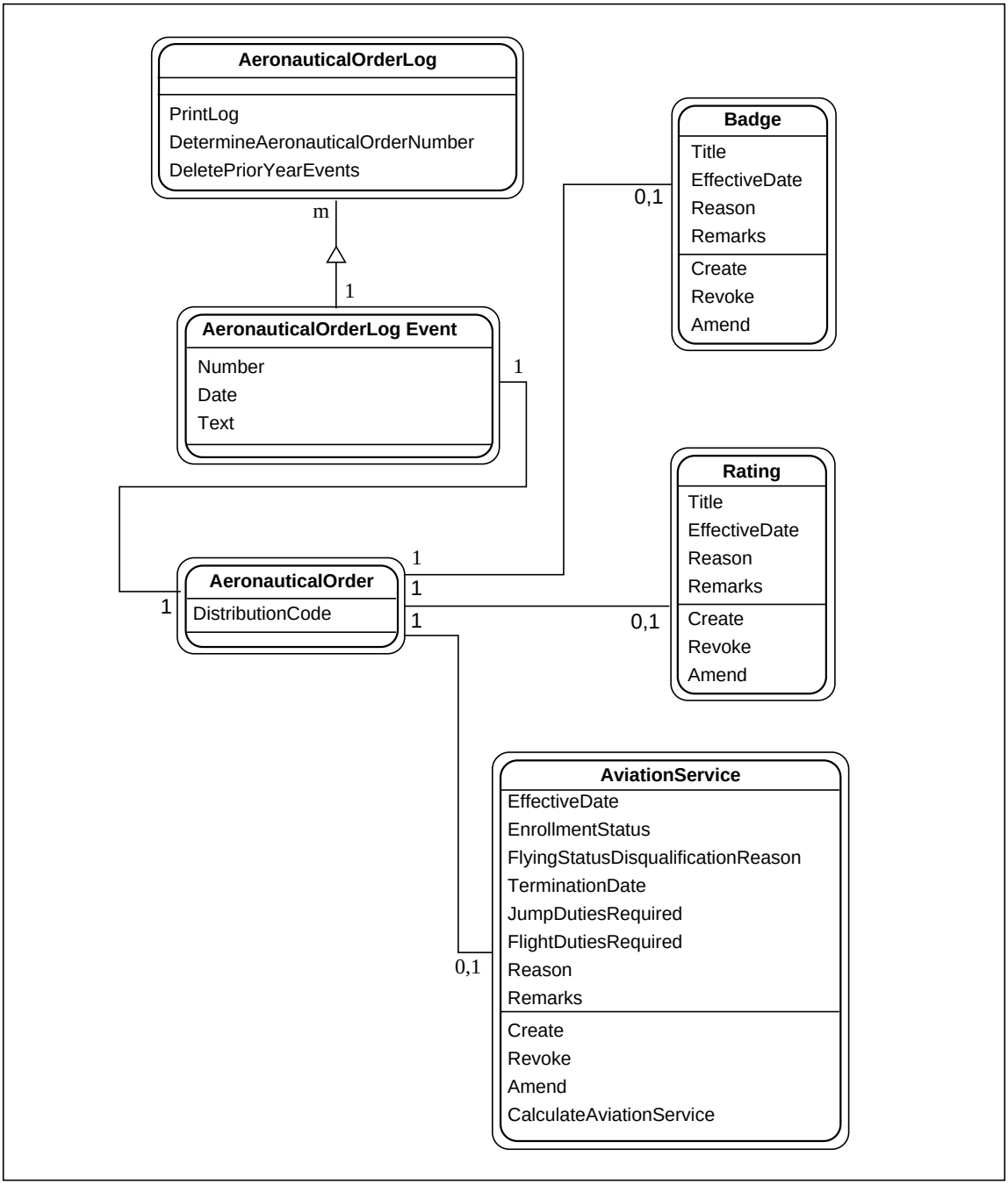



**Figure 27. OO Analysis Model**

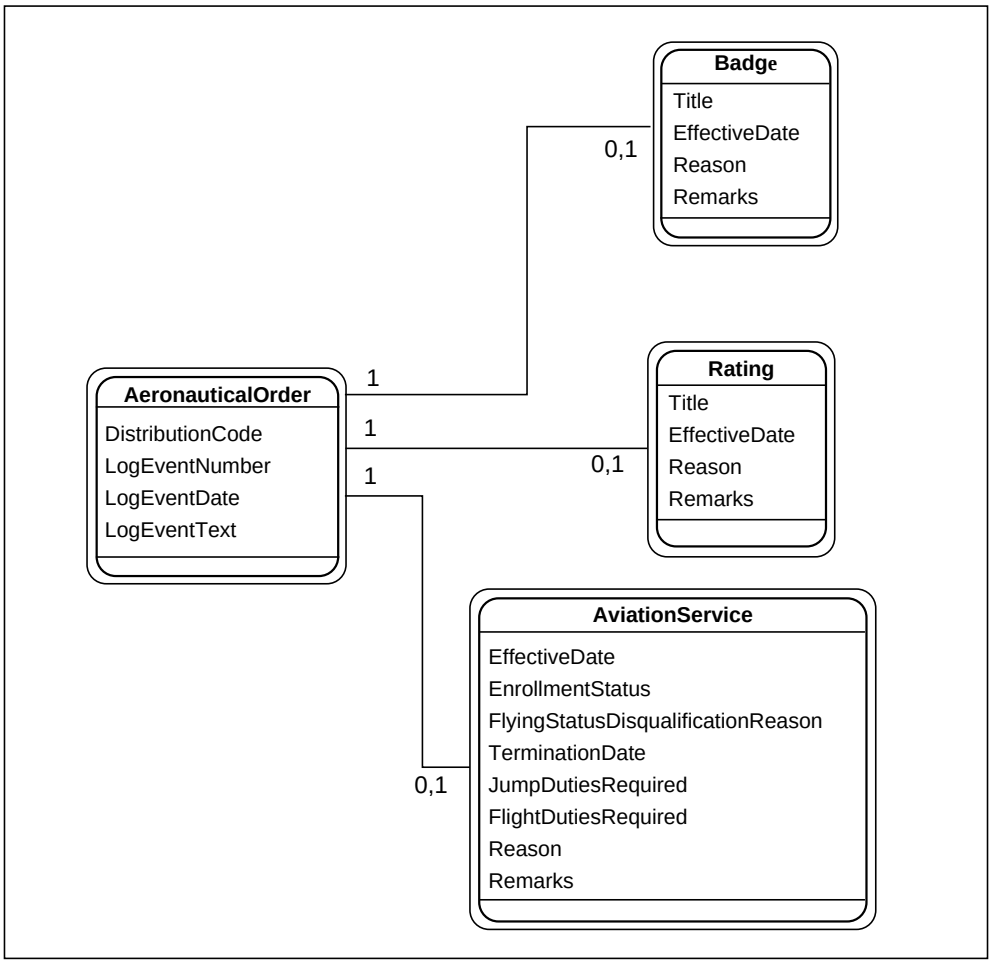



**Figure 28. Information Model**

### 3.10.2 Software Requirements Specification

One of the major outputs of the Software Analysis phase is the formal Software Requirements Specification (SRS). The bulk of the AFORMS SRS consists of tailored DIDs for the objects in the AFORMS OO analysis model. To illustrate their format, members of a small subset of these DIDs taken from the OO analysis model segment depicted in Figure 28 are presented in the following subsections. In addition, we include illustrative portions of other essential aspects of the AFORMS SRS.

**Note:** The following implicit operations are stated once for the model and apply to all objects in the model. They will not be repeated under an object in the model unless the operation is specialized for that object.

## Implicit Operations

- Create: This service creates and initializes a new object in a class.
- Connect: This service associates or disassociates one object to another.
- Access: This service gets or sets the attribute value of an object.
- Release: This service disconnects and deletes an object.

## Requirements Object Model

**AeronauticalOrder (KV-SRS-C-0004)**

Description: This object represents an Air Force order that places an individual on flying status, changes Aviation Service Codes, terminates aviation service, and awards aeronautical ratings.

Instances: Flyer.

Attributes:

DistributionCode (KV-SRS-A-0823): This attribute represents the necessary distribution as defined by the StandardReport.

Data Dictionary Name: Aeronautical Order Distribution Code.

Authority: AFR 60-13.

Reason (KV-SRS-A-0824): This attribute represents the reason causing the AeronauticalOrder action.

Data Dictionary Name: Aeronautical Order Reason Text.

Authority: AFR 60-13.

TextBody (KV-SRS-A-0825): This attribute represents the text information necessary for the Aeronautical Order's body.

Authority: AFR 60-13.

Services:

RequestAOPublication (KV-SRS-S-0031): This service requests publication of an aeronautical order IAW AFR 10-7, Chapter 4, Figure 4-1. Objects of this class will be deleted following successful print action and setting of the Flyer.CBPOFlag.

Software Requirements:

a. Activate AeronauticalOrderLog.DetermineAONumber passing parameters for AeronauticalOrderLogEvent.Text.

b. Message StandardReport to print AeronauticalOrder.

c. Following successful print action, set Flyer.CBPOFlag.

d. Message class to delete this object.

SSS Requirements Satisfied: SSS-0114, SSS-0111, SSS-0157, SSS-0091.

**AviationService (KV-SRS-C-0008)**

Description: This object provides information regarding pay entitlements and current flying status.

Instances: AviationServiceHistory.

Attributes:

EffectiveDate (KV-SRS-A-0071): This attribute represents the effective date of the Aviation Service Code. It can be established by the effective date of duty establishing the aviation service described by the assigned code.

Data Dictionary Name: Aviation Service Effective Date.

Authority: DoD Directive 8320.1.

EntitlementStatus (KV-SRS-A-0081): This attribute represents the individual's entitlement status for aviation service.

Data Dictionary Name: Aviation Service Aeronautical Order Rated Entitlement Status Code.

Authority: AFR 60-1, Chapter 2, Figure 2-2.

FlightDutiesRequired (KV-SRS-A-0041): This attribute identifies the flight duties that are authorized per IAW AFR 10-7, Chapter 4 (Required to Perform Frequent and Regular Flight).

Data Dictionary Name: Aviation Service Aeronautical Order Flight Duties Required Code.

Authority: AFR 10-7, Figure 4-1, item 4.

FlyingStatusDisqualificationReason (KV-SRS-A-0082): This attribute represents the flying status of the individual or the reason for inactive status.

Data Dictionary Name: Aviation Service Disqualification Reason Code.

Authority: AFR 60-1, Chapter 2, Figure 2-2.

JumpDutiesRequired (KV-SRS-A-0040): This attribute identifies that jump duties are authorized (Required to Perform Parachute Jump Duties).

Data Dictionary Name: Aviation Service Aeronautical Order Jump Duties Required Code.

Authority: AFR 10-7, Figure 4-1, item 5.

TerminationDate (KV-SRS-A-0072): This attribute represents the termination date of the Aviation Service Code. This is the last day the AO will be effective.

Data Dictionary Name: Aviation Service Termination Date.

Authority: AFR 60-1, Chapter 2, Paragraph 2-4, and Figure 2-2, 2-3.

Services:

Amend (KV-SRS-S-0825): This service will amend objects of this class and create an instance of an Aeronautical Order.

Software Requirements:

a. Amend existing AviationService.

b. Determine distribution for AeronauticalOrder.

c. Activate AeronauticalOrder.Create with parameters for DistributionCode, Reason, and information necessary for the AeronauticalOrder's TextBody.

SSS Requirements Satisfied: SSS-0092.

CalculateAviationService (KV-SRS-S-0081): This service calculates total months on flying status per IAW AFR 60-1, Paragraphs 2-3, 2-4, Figures 2-2, and 2-3.

Software Requirements:

a. If Entitlementstatus is active, calculate and return the number of months from Date Effective and DateTermination.

b. If EntitlementStatus is inactive, return zero.

SSS Requirements Satisfied: SSS-0107

Create (KV-SRS-S-0864): This service will, in addition to the implicit create requirements, create an instance of an AeronauticalOrder after determining its distribution per AW AFR 10-7, Chapter 4, Table 4-1.

Software Requirements:

a. Create object of this class.

b. Determine distribution for AeronauticalOrder.

c. Active AeronauticalOrder.Create with parameters for DistributionCode, Reason, and information necessary for the AeronauticalOrder's TextBody.

SSS Requirements Satisfied: SSS-0091, SSS-1001.

Revoke (KV-SRS-S-0826): This service will revoke objects of this class and create an instance of an AeronauticalOrder.

Software Requirements:

a. Revoke existing AviationService.

b. Determine distribution for AeronauticalOrder.

c. Activate AeronauticalOrder.Create with parameters for DistributionCode, Reason, and information necessary for the AeronauticalOrder's TextBody.

SSS Requirements Satisfied: SSS-0093.

**Badge (KV-SRS-C-0009)**

Description: This object signifies completion of specialized training and qualification to perform a specific air crew skill. Advanced badges generally recognize extended periods of aviation service and experience. Aviation badges can be manually awarded, removed, or modified at the discretion of the user.

Instances: Flyer.

Attributes:

DateBadgeAwarded (KV-SRS-A-0121): This attribute represents the date a badge is awarded.

Data Dictionary Name: Aviation Badge Awarded Date.

Authority: AFR 10-7, AFR 60-13.

Title (KV-SRS-A-0091): This attribute represents the title of the badge.

Data Dictionary Name: Aviation Badge Code.

Authority: AFR 60-13, Table 7-1; AFR 35-5, Paragraph 4.

Services:

Amend (KV-SRS-S-0829): This service will amend objects of this class and create an instance of an AeronauticalOrder.

Software Requirements:

a. Amend existing AviationService.

b. Determine distribution for AeronauticalOrder.

c. Activate AeronauticalOrder.Create with parameters for DistributionCode, Reason, and information necessary for the AeronauticalOrder's TextBody

SSS Requirements Satisfied: SSS-0092.

Create (KV-SRS-S-0831): This service will, in addition to the implicit create requirements, create an instance of an AeronauticalOrder after determining its distribution per IAW AFR 10-7, Chapter 4, Table 4-1.

Software Requirements:

a. Create object of this class.

b. Determine distribution for AeronauticalOrder.

c. Active AeronauticalOrder.Create with parameters for DistributionCode, Reason, and information necessary for the AeronauticalOrder's TextBody.

SSS Requirements Satisfied: SSS-0091, SSS-1001.

Revoke (KV-SRS-S-0827): This service will revoke objects of this class and create an instance of an AeronauticalOrder.

Software Requirements:

a. Revoke existing AviationService.

b. Determine distribution for AeronauticalOrder.

c. Activate AeronauticalOrder.Create with parameters for DistributionCode, Reason, and information necessary for the AeronauticalOrder's TextBody.

SSS Requirements Satisfied: SSS-0093.

**Instance Connections**

This section describes the instance connections in AFORMS. An instance connection identifies an association between two objects. The range entry identifies how many instances of an object may be associated with another object. For example, in Figure 29, ObjectOne can be associated with zero to many ObjectTwos; however, ObjectTwo can be associated with only one ObjectOne.


**Figure 29. Cardinality Example**

Selected instance connections from our examples of AFORMS could be documented, as shown in the following sections.

**Additional Requirements**

*Sizing and Timing Requirements*. The database size for the new AFORMS will range from 30 Mb to 200 Mb, depending on the organization being supported. The resources required of both memory and the central processing unit (CPU) will be addressed when the target platform is selected.

*Security Requirements*. The data manipulated by AFORMS is SENSITIVE—UNCLASSIFIED. The system supports the use of the Privacy Act and For Official Use Only security caveats. The target system must comply with minimum protection of the C2 level of trust as defined in DOD 5200.28-STD, *Trusted Computer System Evaluation Criteria (TCSEC)* [DOD85]*)*. The classification is UNCLASSIFIED, but AFORMS will support the Privacy Act of 1974 and for Official Use Only information. The association of the individual's name with a social security number, home address, home phone number, and other personally sensitive data must be controlled under the provisions of the Privacy Act. AFORMS is used to track the

unit's air crew training program and flying accomplishments of the unit and all assigned air crew members.

*Design Constraints*. AFORMS shall accommodate the following design constraints:

a. Target system must support transaction histories for all transactions that cause the database to be modified.

b. Target system must support Error Management Reports for database discrepancies.

c. Target system must provide multi-tasking function capabilities.

d. Target system must provide the capability to transmit external electronic mail from the application.

e. Target system must provide the ability for the system administrator to select the communication method (air gap or electronic), for each external interface when the software is installed.

f. AFORMS will be developed to conform and support various modes of operation (regional center, Air Force base, Air National Guard, Reserve Units, and deployable operation), and to support 1 to 1,000 wings.

g. Target platform must be able to support COTS products.

h. Software products will be designed both to make maximum, feasible use of existing software products, and to develop software products for subsequent reuse to the maximum, feasible extent.

AFORMS will be used on open systems specified by the Government and should be portable to the open systems that have the following features:

a. An Ada compiler compliant with ANSI/MIL-STD-1815A-1983 [ANSI83].

b. A C compiler that produces executable binaries for the target platform. (The C compiler must produce object code that can be linked into an Ada executable

c. ANSI SQL that supports dynamic SQL.

d. A version of the Xlib/Xt/Motif libraries must be supplied for the target platform (X Window system).

e. A version of the Ada X Interface (AXI) must be provided for the platform (X Window

## 3.11 PRELIMINARY DESIGN

The major activities of this phase are illustrated in Figure 30 on page 63.

*Logical Database Design*. The logical database design developed during this phase is represented by a logical database model. A small portion of this model corresponding to the OO analysis and Information Models, previously depicted in Figure 27 and Figure 28, is illustrated in Figure 31 on page 64. The logical database design model establishes the structuring of data, represented by tables and relationships. The logical database design model is developed from the Information Model and consists of the following activities.

*Evaluate Interdependencies of Data Objects*. Evaluate the interdependencies or relationships between the data objects on the Information Model. Identify the pieces of data that need to be present in each data object to represent each of the relationships. Identify candidate logical identifiers for each object.

*Determine Logical Identifiers*. From the candidate logical identifiers, pick a single logical identifier that best determines uniqueness for the data object. Considerations in choosing the logical identifier include size, integrity, and need to change. Smaller candidate logical identifiers with a higher degree of integrity are generally better logical identifiers. Sometimes a candidate logical identifier may have the need to change. Changing candidate logical identifiers should not be used as logical identifiers unless it is absolutely necessary. If changing logical identifiers are used, they need to be noted so that they can be reevaluated during the physical design stage.

*Validate the Logical Database Design Model*. After the logical database design model is stable, validate it against the OO analysis and Information models to ensure that all of the functionality is present in the logical database design model. Throughout the logical design of the database, keep in mind concurrent access requirements and data integrity.

*Build a CRUD Matrix*. A Create, Read, Update, and Delete (CRUD) matrix is used to determine how the database is accessed by the application. It also helps to identify the critical transactions for the database. Build a CRUD matrix to evaluate how the data objects are used in the screens, in reports, over the external interfaces, and for *ad hoc* queries.

*Perform Logical Costing Analysis*. Logical costing is a predetermined method of identifying potentially costly access requirements prior to physically designing and implementing the database.

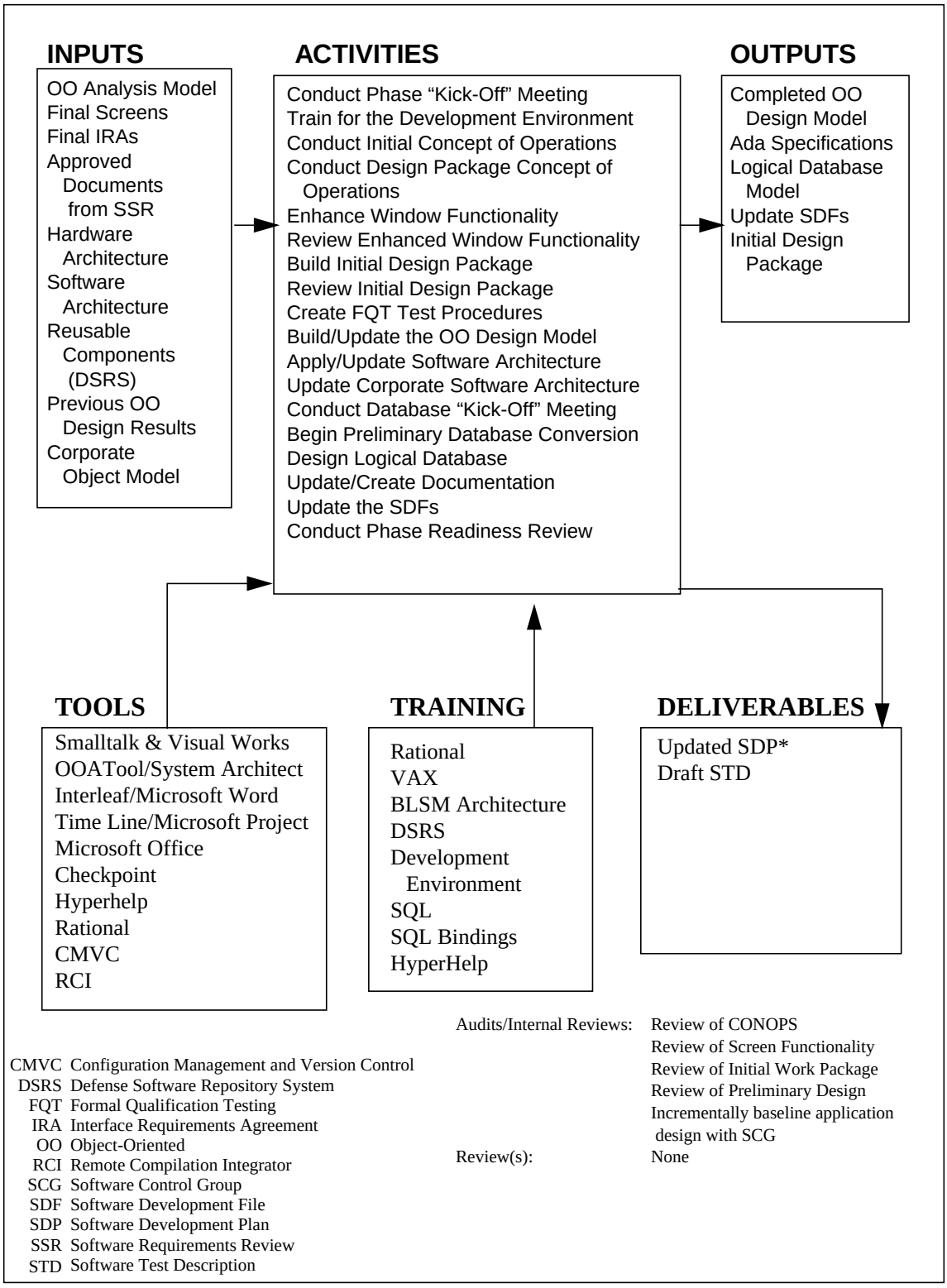



**Figure 30. Preliminary Design**

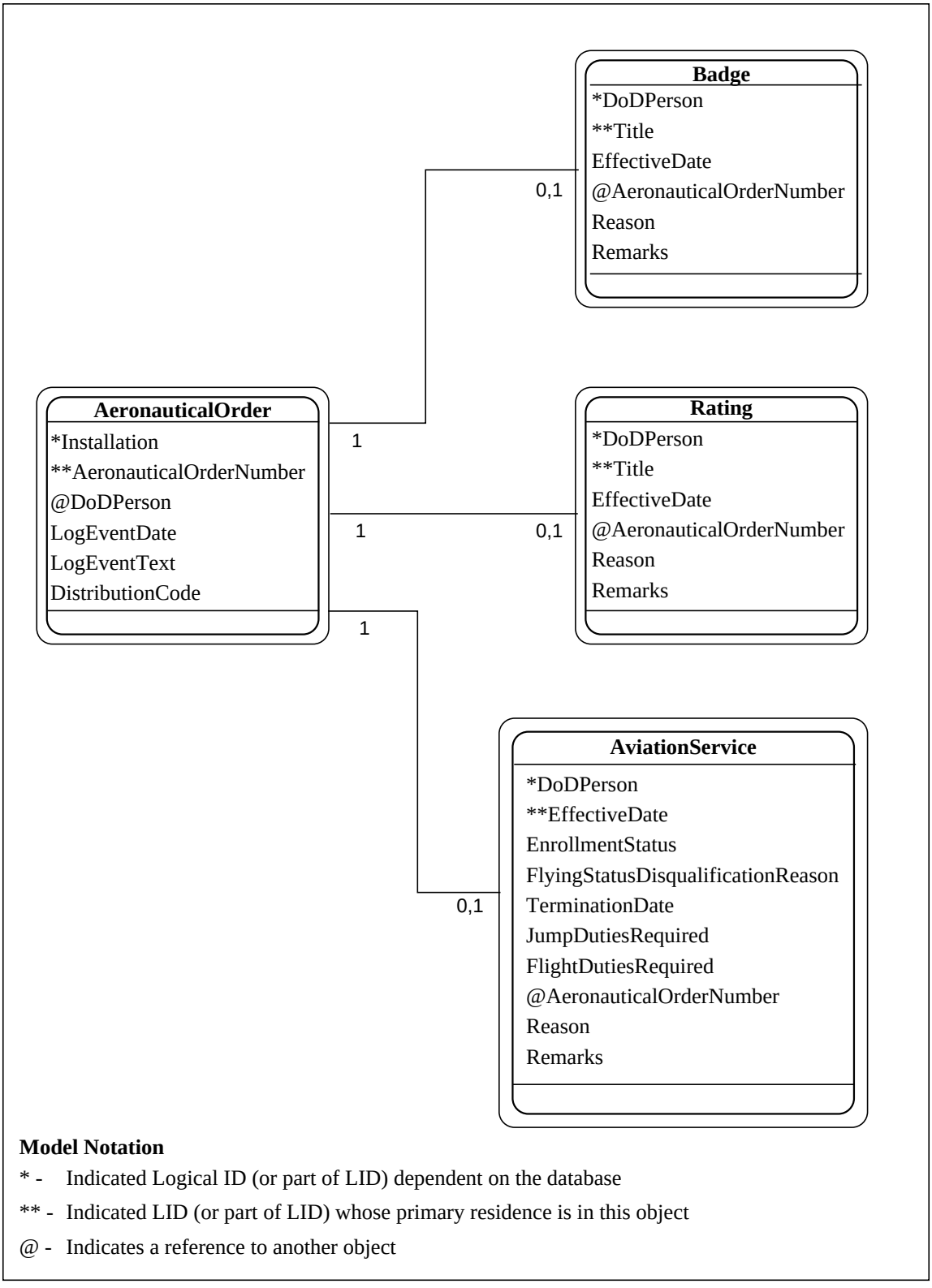



**Figure 31. Logical Database Model**

First, all tables are given a value representing the “average” expected population for the given database. Next, each requirement identified in the System Requirements Specification, which uses the database, is “costed” in terms of (1) the adds, changes, deletes across the tables it must touch to fulfill the requirements; and (2) the processing frequency of the requirement. A total cost for the requirement is calculated.

After the requirements are costed, the “high cost” ones are evaluated to determine what steps (such as adding a secondary index or removal of foreign keys) can be taken to reduce the cost. Each high-cost requirement is re-costed using the readjusted table structures, and the total logical costing is calculated again. Trade-offs are then evaluated (high cost vs. processing frequency) and documented.

*Review the Logical Database Design Model*. At the end of the logical design stage, an Internal Review is held. The task analysts, designers, and functional users review the logical database design for functionality. Comments are collected and incorporated. At this point the logical database design is considered informally baselined.

*Handle the PRs Received During the Logical Design Stage*. Problem Reports (PRs) approved during the logical design stage are directly incorporated into the logical database design model.

## 3.12 DETAILED DESIGN

Beginning with the Detailed Design phase, three documents are drafted incrementally to support the incremental and iterative development methodology: the Software Design Document (SDD), which is incrementally baselined; the Requirements Tracking Matrix (RTM); and the software test cases within the Software Test Description (STD).

*Physical Database Design and Implementation*. Using the logical database and the OO design model designed in the previous phase, the database specialist both designs and creates the physical database for the modernizing system. Figure 32 on page 66 depicts the physical database design which consists of the following steps.

*Develop the Physical Database Model*. The physical database design model is developed from the logical database design model using a CASE tool The CASE tool produces a physical database design model as shown in the following figure.

*Create Tables*. The SQL Data Definition Language (DDL) statements used to create the database are generated from the physical database design model using the SQL type and size

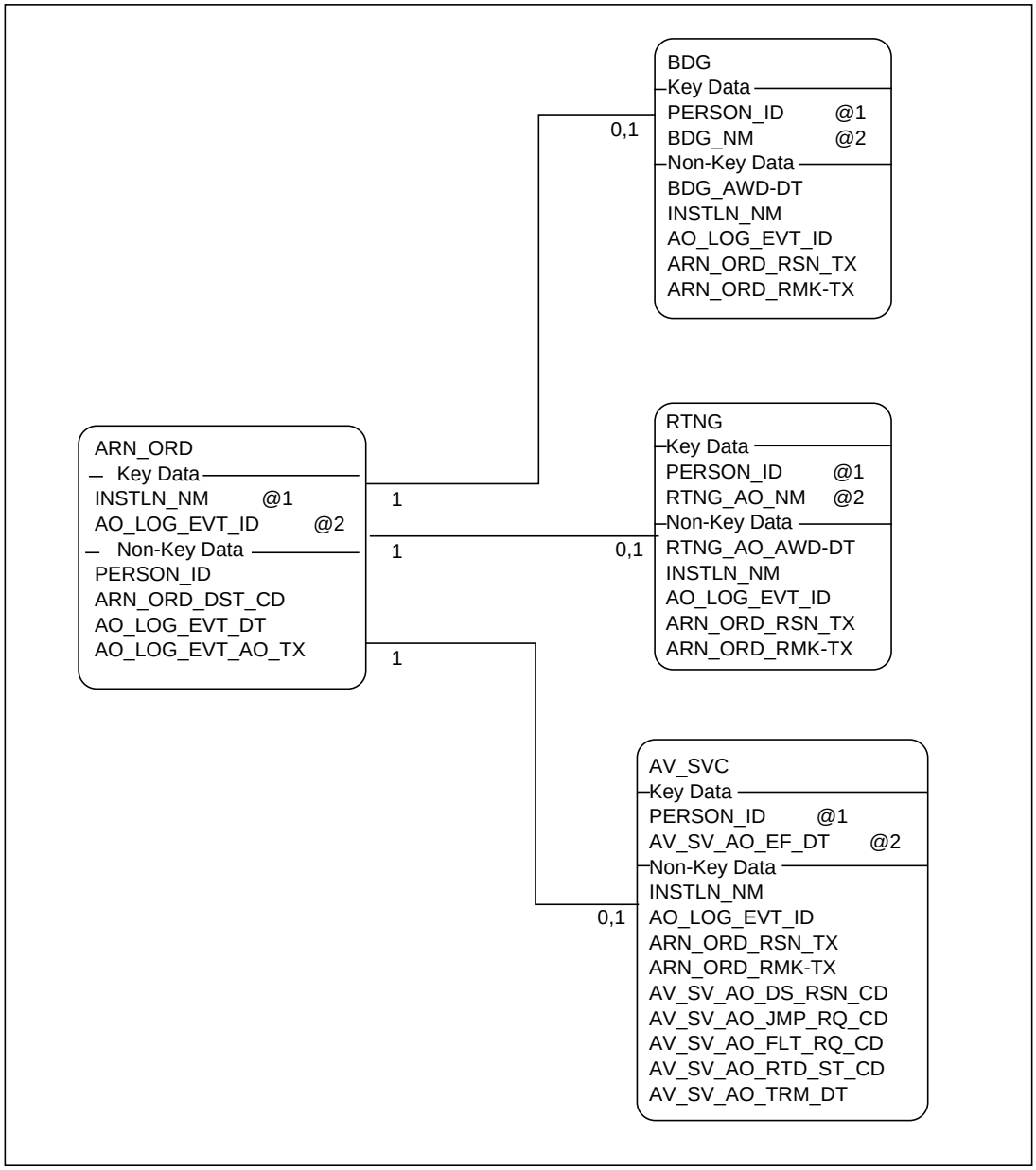



**Figure 32. Physical Database Model**

information entered behind the model. NOT NULL and DEFAULT constraints are added for each column that requires them.

*Group Tables*. Tables are analyzed and grouped according to their usage and relationships to one another. Ease of recovery also guides the grouping of tables.

*Size Tables*. Each table's size is calculated based upon system variables (block and word sizes), expected population, and RDBMS table overhead. Table sizing is calculated using the RDBMS specific formulas found in the RDBMS documentation.

*Create Referential Integrity Constraints*. SQL statements used to impose referential integrity constraints on each table are created. The PRIMARY and FOREIGN KEYs are identified and constraints are written for them.

*Create Edits and/or Validation Constraints*. The SQL statements used to impose edits and/or validation constraints on the columns in each table are created. Where it is most efficient for the database to handle the edits, a CHECK constraint is written for the column.

*Build Support Command Files*. Command files needed to support the database operation are developed. These files are used to build and delete the database. Preliminary recovery and/or backup procedures and command files are also developed.

*Build Views*. Creation of table views that provide limited access to specific data within a table or tables will be developed based upon security considerations. Views are also built to represent user and Corporate Object Model (COM) objects.

*Generate Database Design Document*. The Database Design Document (DBDD) is the formal documentation of the physical database design. It includes the physical database design model, all of the table definitions, and all Standard Data Element (SDE) information. The SDE information includes table name, data dictionary name, database access name, size, type, and data edits.

*Baselining the Physical Database Design*. When the physical database design is considered to be stable (i.e., validated against screens, reports, and external interfaces), a peer review is held with other data administration staff. After the peer review comments have been incorporated, an Internal Review (IR) is held to assess the physical database design. After the physical database design is accepted by the attendees of the IR, the physical design model can then be baselined by the Software Control Group.

## 3.13 CODE AND TEST

In the Code and Test Phase, depicted in Figure 33, the Ada bodies are built from the Ada specifications developed during the Detailed Design phase, and the individual software components undergo testing by the developer. Once the software components have been unit tested by the developers, they undergo further testing by the contractor's Independent Test Group.

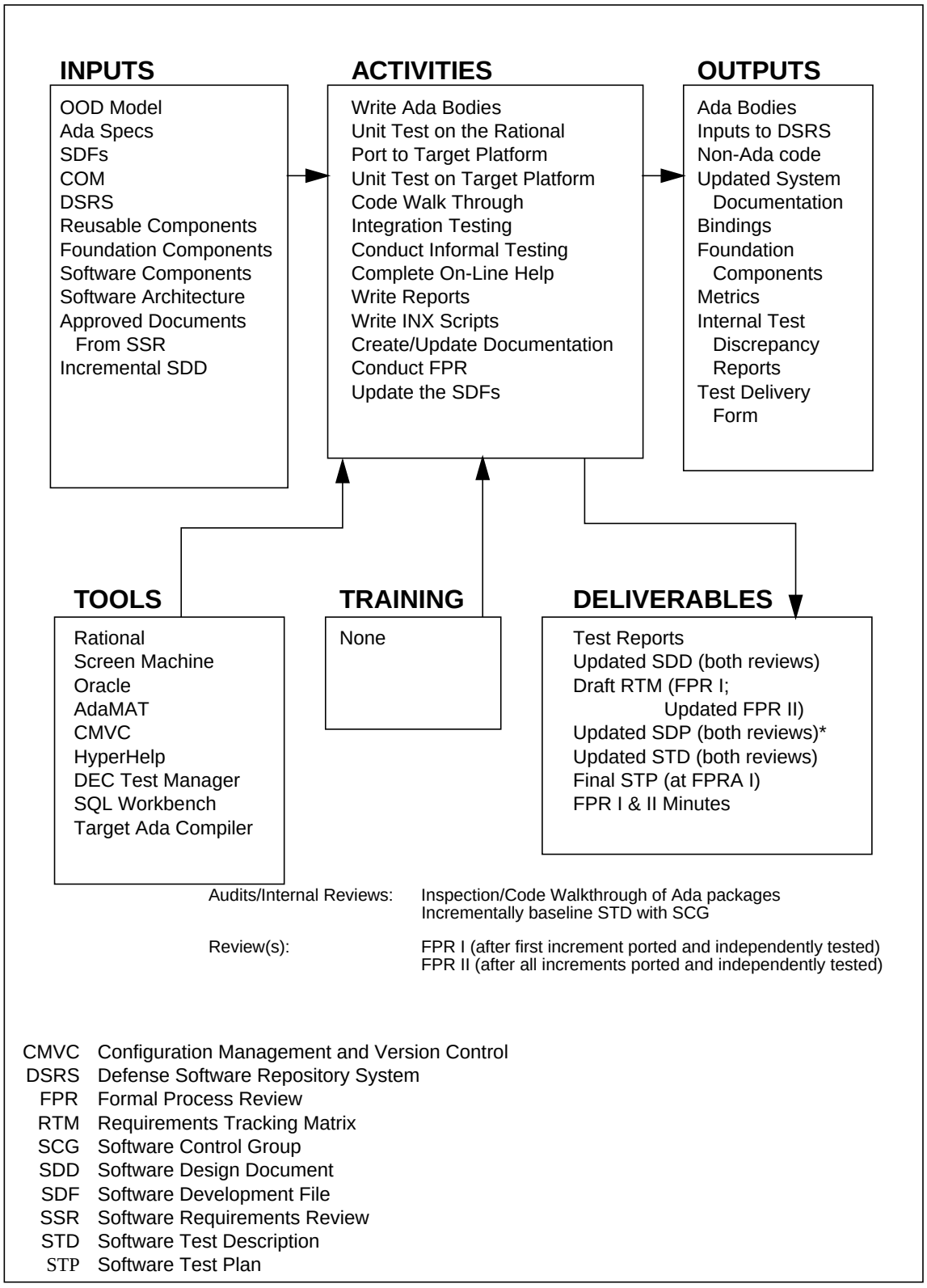



**Figure 33. Code and Test**

One of the tools used during the coding of the Ada bodies is AdaMAT. It is used to assess the overall quality of the software and helps identify any difficulty in maintaining and porting the software.

The Task Leader is responsible for ensuring that all Ada bodies and non-Ada code are inspected. The Task Leader also ensures that any code which falls below the AdaMAT thresholds and any high-risk increments have a code walk-through performed. The unit test cases and AdaMAT results are reviewed during the code walk-through.

Use of the DSRS is made during this phase as software engineers query the system for foundation components (common utilities), reusable bindings, and other software components.

In the incremental software development process, Formal Progress Review I (FPR I) is held after detailed design and code and test of the first increment have been completed. FPR II is held after all increments have completed detailed design and code and test

## 3.14 FORMAL SOFTWARE TESTING

During the Formal Software Testing phase, as illustrated in Figure 34 on page 70, the Independent Systems Test Group tests the total system against the Software Test Plan (STP). The DEC Test Manager is a tool used during this phase to organize tests, select tests for execution, and review and verify test results. DEC Test Manager also automates the regression testing process. In addition, the system is tested on the target hardware platform.

All remaining system documentation, with the exceptions of the Lessons Learned (LL), Software Test Report (STR), and the Software Product Specification (SPS), is finalized prior to the Test Readiness Review (TRR). The most important deliverable of the entire development process, the Computer Software Product End Item (CSPEI), is delivered at the TRR.

The TRR is held to review test results, system documentation, and the formal procedures used in the test of the system. The successful completion of the TRR indicates that the software products are ready for Qualification Test and Evaluation (QT&E). The final function point count is presented at TRR**.**

## 3.15 QT&E SUPPORT

The major activities of Qualification Test and Evaluation (QT&E) support are depicted in Figure 35 on page 71. The final STR, SPS, and the final input to the task-specific Lessons Learned (LL) are delivered to the customer for approval.

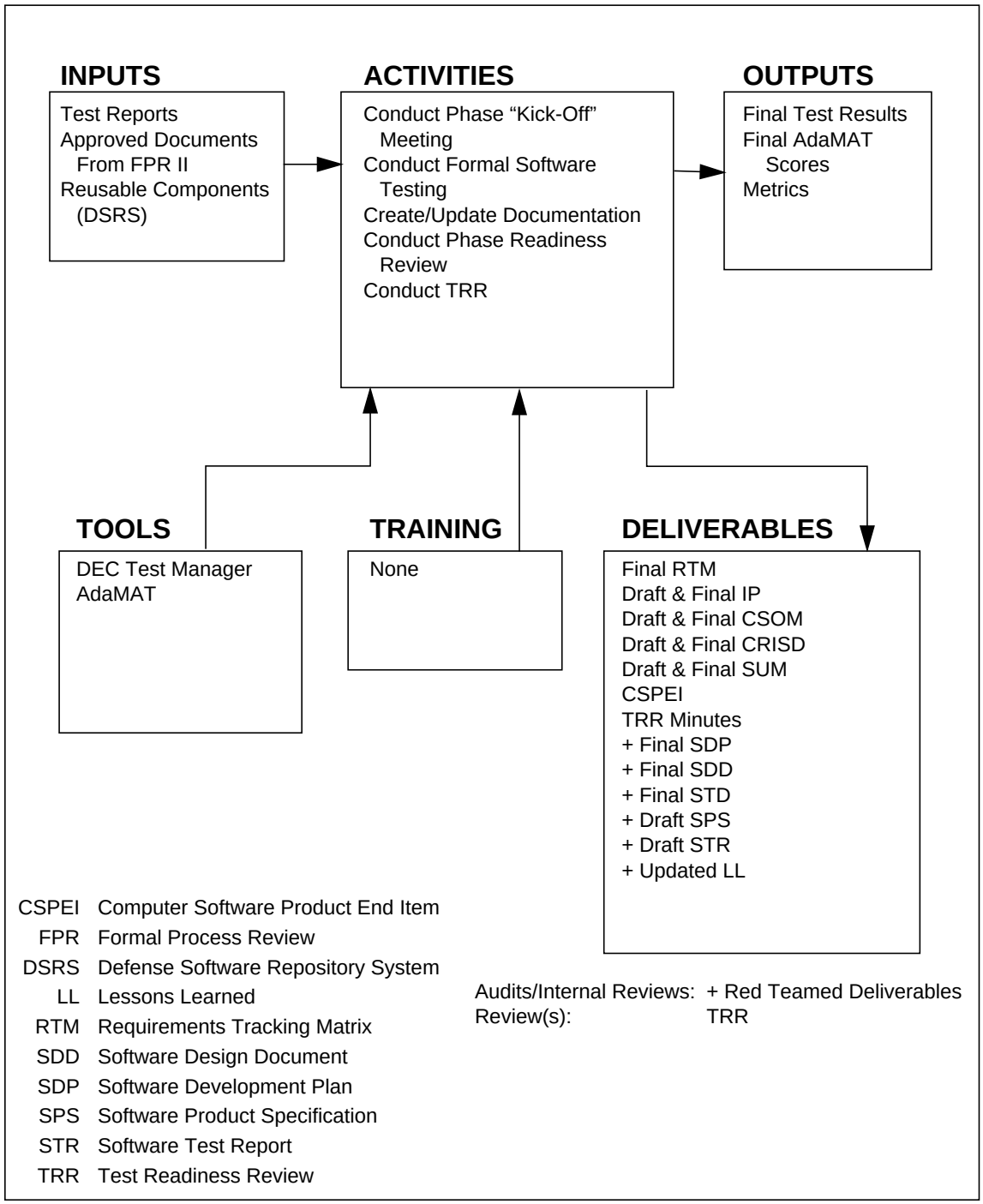



**Figure 34. Formal Software Testing**

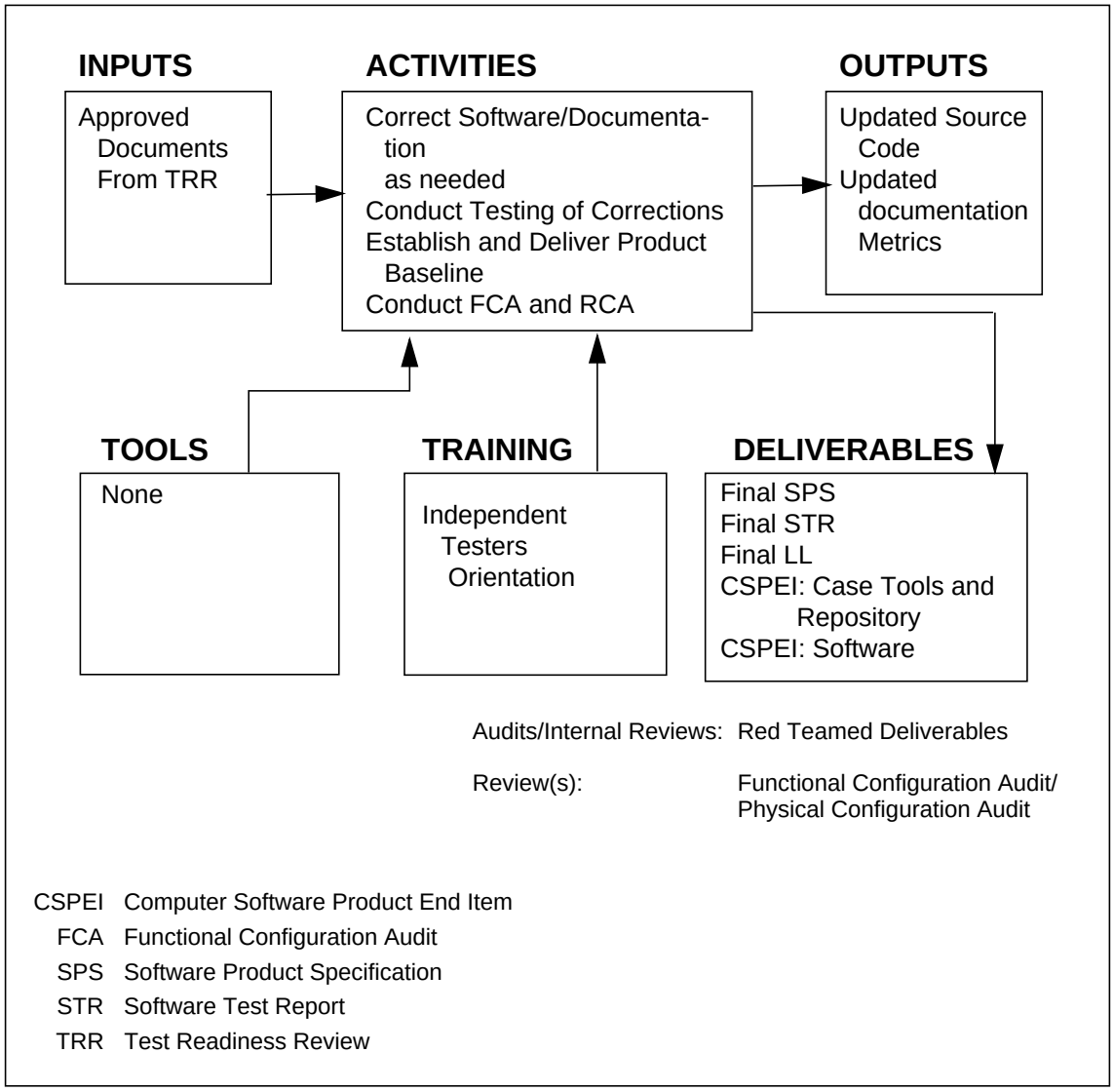



**Figure 35. QT&E Support**

# CHAPTER 4. SUMMARY OF GUIDELINES AND ISSUES

## 4.1 OOT REENGINEERING GUIDELINES

In the course of investigating strategies and tactics for reengineering, we have collected a number of guidelines for choosing a particular strategy and tactics and applying them in a variety of contexts. Reengineering with OOT ordinarily requires considerable effort for transforming any legacy models for analysis and design since they are unlikely to be object oriented nor will they map directly into object-oriented models. Thus, the demands of these efforts must be accommodated in any plan to reengineer using OOT and may limit the scope of the reengineering effort that is feasible at any particular stage of migration.

DoD policy requirements [DOD92] that functional (business) process improvement activities precede any systems reengineering activity further increase the demands of performing reengineering with OOT. This is a special problem for OOT use because the DoD-directed analysis techniques for these activities, using IDEF0 and IDEF1X, create functional models that do not have a direct mapping into OO models. Chapter 2 identifies guidelines on how to use IDEF0 models to generate scenarios, or use cases, that could provide a basis for subsequent OO development. IDEX1X models are also identified as potential sources of entities and other static aspects of object models.

The other principal guidelines offered on reengineering describe how to identify potential features for OO modeling from the structured analysis models of legacy systems when such models exist. General rules for extracting object models from data flow models include the following:

- Terminators will generally map to class or object in the problem domain.
- Data stores will map to class or objects.
- Data flows can correspond to classes, objects, or attributes.
- Control flows often correspond to specific events.
- Processes may correspond to services or operations within a class.

In addition, a more involved procedure (Bailin's Object-Oriented Specification) is described that uses both entity-relationship models and data flow models to help build object models [BAI89].

## 4.2 OOT REENGINEERING ISSUES

To summarize, this investigation of strategies for reengineering DoD software systems using OOT has uncovered two major issues regarding systems reengineering.

- **Non-OO specifications in legacy systems.** It is very likely that the artifacts (requirements, design, and database specifications) obtained from a legacy system will not be in an OO form. Most older systems were built before OO or even structured techniques were in common use. As a result, it may not be a simple issue to forward engineer a new system using these non-OO specifications.
- **Non-OO functional process improvement policy.** Current DoD policy requires that a functional process improvement activity be carried out before reengineering an information system, but the techniques and models to support functional process improvement are not object oriented. These functional process improvement models are supposed to be used as input to the information system reengineering or development.

In both cases there is a paradigm shift required from either the system artifacts or functional process improvement models. Chapter 2 offers strategies to use these extant models; however, the paradigm shift will not be automatic and the building of an OO system will still require substantial effort by system developers to construct OO specifications and models.

# LIST OF REFERENCES


[AIK94] P. Aiken et al., "DoD Legacy Systems—Reverse Engineering Data Requirements," *Communications of the ACM*, Vol. 37, No. 5, May 1994.

[ANSI83] American National Standards Institute, ANSI/MIL-STD-1815A-1983, *Ada Programming Language*, New York, NY, 1983.

[ARN93] Arnold, R., ed., *Software Reengineering*, IEEE Computer Society Press, 1993.

[BAI89] Bailin, S.C. "An Object-Oriented Requirements Specification Method," Communications of the ACM, Vol. 32, No. 5, May 1989, pp. 608-623.

[BLSM93] *Base-Level System Modernization (BLSM), A Strategy for the Future*, prepared by Standard Systems Center/XON, Maxwell AFB, Gunter Annex, AL, November 1, 1993.

[BOO94] G. Booch, *Object-Oriented Analysis and Design with Applications*, The Benjamin/Cummings Publishing Company, Inc., Redwood City, CA, 1994.

[BRU92] T. A. Bruce, *Designing Quality Databases with IDEF1X Information Models*, Dorset House Publishers, New York, NY, 1992.

[CDYD91] P. Coad and E. Yourdon, *Object-Oriented Analysis*, 2nd edition, Yourdon Press, Englewood Cliffs, NJ, 1991.

[CIM93a] Center for Information Management, *Automated Information Systems Software Reengineering Risks Taxonomy Report*, Defense Information Systems Agency, Joint Interoperability Engineering Organization, September 1993.

[CIM93b] Center for Information Management, *Information System Criteria for Applying Software Reengineering: Guidelines for Identifying Candidate Information Systems for Software Reengineering*, Defense Information Systems Agency, May 1993.

[CIM94] Center for Information Management, *Center for Information Management Software Systems Reengineering Process Model, Version 2.0*, draft, Defense Information Systems Agency, Joint Interoperability Engineering Organization, September 1994.

[DEM79] T. DeMarco, *Structured Analysis and System Specification*, Prentice-Hall, Englewood Cliffs, NJ, 1979.

[DOD92] Department of Defense, DOD 8020.1-M (Draft), *Functional Management Process for Implementing the Information Management Program of the Department of Defense,* August 1992.

[DOD93a] Department of Defense, DOD Directive 8120.1, *Life-Cycle Management (LCM) of Automated Information Systems (AISs)*, January 14, 1993.

[DOD93b] Department of Defense, DOD Instruction 8120.2, *Automated Information System Life-Cycle Management Process, Review, and Milestone Approval Procedures*, January 14, 1993.

[DOD94] Department of Defense, MIL-STD-498, *Software Development and Documentation*, December 5, 1994.

[DOD85] DOD 5200.28-STD, *Trusted Computer System Evaluation Criteria (TCSEC)*, Washington, DC, December 1985.

[HARR93] Harris Data Services Corporation, *Software Development Plan (SDP) for the Base Level System Modernization*, Contract No. F01620-88-D-0086, CDRL Sequence No. A056. Prepared for Standard Systems Center (AFCC), Director of Contracting, Maxwell AFB, Gunter Annex, AL. Prepared by Harris Data Services Corporation, Montgomery AL, October 1993.

[HAT87] D. Hatley and I. Pirbhai, *Strategies for Real-Time System Specification*, Dorset House, New York, 1987.

[HUTT94] A. T. F. Hutt, ed., *Object Analysis and Design: Description of Methods,* John Wiley & Sons, Inc., New York, NY, 1994.

[IDA86] Institute for Defense Analyses, *A Descriptive Evaluation of Automated Software Cost-Estimation Models,* Alexandria, VA, October 1986.

[IDA93] Institute for Defense Analyses, *User's Manual for the Functional Economic Analysis Model (Version.3.0)*, IDA Paper P-2904, Alexandria, VA, December 1993.

[IDA95a] Institute for Defense Analyses, *System Reengineering Assessment Method*, IDA Paper P-2904, Alexandria, VA, January 1995.

[IDA95b] B. Haugh, A. Noor, D. Smith, K. Jordan, *Legacy System Wrapping for Department of Defense Information System Modernization*, IDA Paper P-3144, draft, Institute for Defense Analyses, Alexandria, VA, July 1995.

[JACO93] I. Jacobson, M. Christerson, P. Jonsson, and G. Overgaard, *Object-Oriented Software Engineering: A Use Case Driven Approach*, Addison-Wesley Publishing Company, Inc., Reading, MA, 1993.

[JLC93] Joint Logistics Commanders Joint Policy Coordinating Group on Computer Resources Management, "Reengineering Economics Handbook," *Proceedings of First Software Reengineering Workshop—Santa Barbara I*, March 1993.

[NEL91] M. Nelson, "An Object-Oriented Tower of Babel," *OOPS Messenger*, Vol. 2, No. 3, July 1991.

[NIE88] K. Nielsen and K. Shumate, *Designing Large Real-Time Systems with Ada*, New York, McGraw-Hill, 1988

[NIN94] J. Ning, A. Engberts, and W. Kozaczynski, "Automated Support for Legacy Code Understanding," *Communications of the ACM*, Vol. 37, No. 5, May 1994. pp. 50-57.

[PRE94] W. Premerlani and M. Blaha, "An Approach for Reverse Engineering of Relational Databases," *Communications of the ACM*, Vol. 37, No. 5, May 1994.

[RUE93] T. B. Ruegsegger, "IDEF0: Function Modeling for the Object-Oriented," briefing presented at Eleventh Annual National Conference on Ada Technology, Williamburg, VA, March 15, 1993. GTE Federal Systems, Chantilly, VA.

[RUG90] S. Rugaber, "Recognizing Design Decisions in Programs," *IEEE Software*, January 1990.

[RUMB91] J. Rumbaugh, M. Blaha, W. Premerlani, E. Frederick, and W. Lorensen, *Object-Oriented Modeling and Design*, Prentice-Hall, Englewood Cliffs, NJ, 1991.

[STS93a] Software Technology Support Center, *Reengineering Technology Report*, Hill Air Force Base, UT, August 1993.

[STS93b] Software Technology Support Center, *Software Estimation Technology Report*, Hill Air Force Base, UT, March 1993.

[USAF89] U. S. Air Force, *Statement of Operational Need (SON)*, HQ USAF/SC 006-89, December 1989.

[USAF92] U. S. Air Force, *HQ USAF/SC Program Management Directive (PMD)*, 0923(3)/PE 386110, July 1992.

[WEG90] P. Wegner, "Concepts and Paradigms of Object-Oriented Programming," *OOPS Messenger*, Vol. 1, No. 1, August 1990.

[WIRF90] R. Wirfs-Brock, B. Wilkerson, and L. Weiner, *Designing Object-Oriented Software*, Englewood Cliffs, NJ, Prentice-Hall, 1990.

[YOUR89] E. Yourdon, *Modern Structured Analysis*, Prentice-Hall, Englewood Cliffs, NY, 1989.

# GLOSSARY

Words used in the definition of a glossary term and that are defined elsewhere are in **bold**.

**Abstraction** — Abstraction consists of focusing on the essential, inherent aspects of an entity and ignoring its accidental properties [RUMB91].

**AIS Program** — A directed and funded AIS effort, to include all **migration** systems, that is designed to provide a new or improved capability in response to a validated need [DOD93a].

**Architecture** — The organizational structure of a system or **CSCI**, identifying its components, their interfaces, and a concept of execution among them [DOD94].

**Automated Information System (AIS)** — A combination of **computer hardware** and computer **software**, data, and/or telecommunications that performs functions such as collecting, processing, transmitting, and displaying information. Excluded are computer resources, both hardware and software, that are either physically part of, dedicated to, or essential in real time to the mission performance of weapon systems; used for **weapon system** specialized training, simulation, diagnostic test and maintenance, or calibration; or used for research and development of weapon systems [DOD93a]. However, as used here, AISs include systems for C2I, C3I, and C4I, even though they may be essential in real time to mission performance.

**Class** — A class can be defined as a description of similar **object**s, like a template or cookie cutter [NEL91]. The class of an object is the definition or description of those attributes and behaviors of interest.

**Collaboration** — A request from a client to a server in fulfillment of a client's responsibilities [HUTT94, p. 192].

**Commercial-off-the-Shelf (COTS)** — Commercial items that require no unique government modifications or maintenance over the life cycle of the product to meet the needs of the procuring agency [DOD93a].

**Computer Hardware** — Devices capable of accepting and storing computer data, executing a systematic sequence of operations and computer data, or producing control outputs. Such devices can perform substantial interpretation, computation, communication, control, or other logical functions [DOD94].

**Computer Program** — A combination of computer instructions and data definitions that enables **computer hardware** to perform computational or control functions [DOD94].

**Computer Software Configuration Item (CSCI)** — An aggregation of software that satisfies an end use function and is designated for separate configuration management by the acquirer. CSCIs are selected based on tradeoffs among software function, size, host or target computers, developer, support concept, plans for reuse, criticality, interface considerations, [the] need to be separately documented and controlled, and other factors [DOD94].

**Contract** — The list of requests that a client **class** can make of a server class. Both must fulfill the contract: the client by making only those requests the contract specifies, and the server by responding appropriately to those requests [HUTT94, p. 192].

**CRC Cards** — **Class-Responsibility**-Collaborator Cards. CRC cards are pieces of paper divided into three areas: the **class** name and the purpose of the class, the responsibilities of the class, and the collaborators of the class. CRC cards are intended to be used to iteratively simulate different scenarios of using the system to get a better understanding of its nature [HUTT94, p. 192].

**Database** — A collection of related data stored in one or more computerized files in a manner that can be accessed by users or computer programs via a database management system [DOD94].

**Database Management System** — An integrated set of computer programs that provide the capabilities needed to establish, modify, make available, and maintain the integrity of a database [DOD94].

**Encapsulation** — . . . (also **information hiding**) consists of separating the external aspects of an **object**, which are accessible to other objects, from the internal implementation details of the object, which are hidden from other objects [RUMB91]. The act of grouping into a single object both data and the operation that affects that data [WIRF90].

**Framework** — A collection of **class** libraries, generics, design, scenario models, documentation, etc., that serves as a platform to build applications.

**Government-off-the-Shelf (GOTS)** — Products for which the Government owns the data rights, that are authorized to be transferred to other DoD or Government customers, and that require no unique modifications or maintenance over the life cycle of the product [DOD93b].

**Inheritance** — Inheritance is the sharing of attributes and operations among classes based on a hierarchical relationship [RUMB91]. Subclasses of a **class** inherit the operations of the parent class and may add new operations and new instance variables. Inheritance allows us to reuse the behavior of a class in the definition of new classes [WEG90].

**Information Hiding** — Making the internal data and methods inaccessible by separating the external aspects of an **object** from the internal (hidden) implementation details of the object.

**Information System** — See **Automated Information System (AIS)**.

**Legacy System** — Any currently operating automated system that incorporates obsolete computer technology, such as proprietary hardware, closed systems, “stovepipe” design, or obsolete programming languages or **database** systems.

**Life-Cycle Management (LCM)** — A management process, applied throughout the life of an **AIS**, that bases all programmatic decisions on the anticipated mis-

| | |
|---|---|
| | sion-related and economic benefits derived over the life of the AIS [DOD93a]. |
| **Message** | Mechanism by which **object**s in an OO system request **service**s of each other. Sometimes this is used as a synonym for **operation**. |
| **Method** | An operation upon an **object**, defined as part of the declaration of a **class**; all methods are **operation**s, but not all operations are methods [BOO94a]. |
| **Migration** | The transition of support and operations of software functionality from a **legacy system** to a **migration system**. |
| **Migration System** | An existing **AIS**, or a planned and approved AIS, that has been officially designated to support standard processes for a functional activity applicable DoD-wide or DoD Component-wide [DOD93a]. Ordinarily, an AIS that has been designated to assume the functionality of a legacy AIS. |
| **Monomorphism** | A concept in type theory, according to which a name (such as a variable declaration) may only denote **object**s of the same **class** [BOO94a]. |
| **Object** | A combination of state and a set of methods that explicitly embodies an **abstraction** characterized by the behavior of relevant requests. An **object** is an instance of an implementation and an interface. An object models a real-world entity (such as a person, place, thing, or concept), and it is implemented as a computational entity that encapsulates state and operations (internally implemented as data and methods) and responds to requestor services. |
| **Object-Based Programming** | A method of programming in which programs are organized as cooperative collections of objects, each of which represents an instance of some type, and whose types are all members of a hierarchy of types . . . somewhat constrained by the existence of static binding and **monomorphism** [BOO94a]. |

.

| | |
|---|---|
| **Object-Oriented Analysis** | A method of analysis in which requirements are examined from the perspective of the classes and objects found in the vocabulary of the problem domain [BOO94a]. |
| **Object-Oriented Decomposition** | The process of breaking a system into parts, each of which represents some **class** or **object** from the problem domain [BOO94a]. |
| **Object-Oriented Design** | A method of design encompassing the process of OO decomposition and a notation for depicting both logical and physical as well as static and dynamic models of the system under design [BOO94a]. |
| **Object-Oriented Programming** | A method of implementation in which programs are organized as cooperative collections of objects, each of which represents an instance of some **class**, and whose classes are members of a hierarchy of classes united via **inheritance** relationships. In such programs, classes are generally viewed as static, whereas objects typically have a much more dynamic nature, which is encouraged by the existence of dynamic binding and **polymorphism** [BOO94a]. |
| **Object-Oriented Technology (OOT)** | OOT consists of a set of methodologies and tools for developing and maintaining software systems using software objects composed of encapsulated data and operations as the central paradigm. |
| **Object Request Broker (ORB)** | Program that provides a location and implementation-independent mechanism for passing a **message** from one **object** to another. |
| **Operation** | Some action that one object performs upon another in order to elicit a reaction [BOO91]. A Service is a specific behavior that an Object is responsible for exhibiting [CDYD91]. |
| **Polymorphism** | The same operation may behave differently on different classes [RUMB91]. |
| **Reengineering** | The process of examining and altering an existing system to reconstitute it in a new form. May include reverse engineering |

(analyzing a system and producing a representation at a higher level of **abstraction**, such as design from code), restructuring (transforming a system from one representation to another at the same level of abstraction), redocumentation (analyzing a system and producing user or support documentation), forward engineering (using software products derived from an existing system, together with new requirements, to produce a new system), retargeting (transforming a system to install it on a different target system), and translation (transforming source code from one language to another or from one version of a language to another) [DOD94].

**Requirement** (1) Characteristic that a system or **CSCI** must possess in order to be acceptable to the acquirer. (2) A mandatory statement in this standard or another portion of the **contract** [DOD94].

**Responsibility** A **contract** that a **class** must support, intended to convey a sense of the purpose of the class and its place in the system [HUTT94, p. 192].

**Service** A service is a specific behavior that an Object is responsible for exhibiting [CDYD91].

**Software** Computer programs and computer databases. **Note:** Although some definitions of software includes documentation, MIL-STD-498 limits the scope of this term to computer programs and computer databases in accordance with Defense Federal Acquisition Regulation Supplement 227.401 [DOD94].

**Software Development** A set of activities that results in **software** products. Software development may include new development, modification, reuse, **reengineering**, or any other activities that result in software products [DOD94].

**Software Engineering** In general usage, a synonym for **software development**. As used in this standard [MIL-STD-498 (DOD94)], a subset of software development consisting of all activities except qualification testing. The standard makes this distinction for the sole purpose of giving separate names to the software engineering

| | |
|---|---|
| | and software test environments [DOD94]. |
| **Software Engineering Environment** | The facilities, hardware, software, firmware, procedures, and documentation needed to perform **software engineering**. Elements may include but are not limited to computer-aided software engineering (CASE) tools, compilers, assemblers, linkers, loaders, operating systems, debuggers, simulators, emulators, documentation tools, and **database** management systems [DOD94]. |
| **Software System** | A system consisting solely of software and possibly the computer equipment on which the software operates [DOD94]. |
| **Weapon System** | Items that can be used directly by the Armed Forces to carry out combat missions and that cost more than 100,000 dollars or for which the eventual total procurement cost is more than 10 million dollars. That term does not include commercial items sold in substantial quantities to the general public (Section 2403 of 10 U.S.C., reference (bb) [DOD93a]. |

# LIST OF ACRONYMS

| | |
|---|---|
| ACIA | Aviation Career Incentive Act |
| ACIP | Aviation Career Incentive Pay |
| AFORMS | Air Force Operations Resource Management System |
| AFR | Air Force Regulation |
| AIS | Automated Information Systems |
| AL | Alabama |
| AMF | Aircraft Maintenance Facility |
| AMS | Aircraft Maintenance System |
| ANSI | American National Standards Institute |
| AXI | Ada X Interface |
| BCA | Business Case Analysis |
| BLSM | Base-Level System Modernization |
| BPI | Business Process Improvement |
| CASE | Computer-Aided Software Engineering |
| CDRL | Contract Data Requirements List |
| CIM | Center for Information Management; Corporate Information Management |
| CM | Configuration Management |
| CMS | Code Management System |
| CMVC | Configuration Management and Version Control |
| COM | Corporate Object Model |
| CONOPS | Concept of Operations |
| COTS | Commercial off the Shelf |
| CPU | Central Processing Unit |
| CRC | Class-Responsibility-Collaborators |
| CRUD | Create, Read, Update, and Delete |
| CSPEI | Computer Software Product End Item |
| DBA | Database Administrator |
| DBDD | Database Design Document |
| DBMS | Database Management Systems |

| | |
|---|---|
| DCPS | Defense Civilian Pay System |
| DDL | Data Definition Language |
| DDRS | Defense Data Repository System |
| DEC | Digital Equipment Corporation |
| DFD | Data Flow Diagram |
| DID | Data Item Description |
| DISA | Defense Information Systems Agency |
| DoD | Department of Defense |
| DODD | Department of Defense Directive |
| DODI | Department of Defense Instruction |
| DOS | Distributed Operating System |
| DSRS | Defense Software Repository System |
| DTM | Digital Equipment Corporation Test Manager |
| E-R | Entity-Relationship |
| FCA | Functional Configuration Audit |
| FPR | Formal Process Review |
| FQT | Formal Qualification Testing |
| GUI | Graphical User Interface |
| IAW | In Accordance With |
| ICOM | Inputs, Controls, Outputs, and Mechanisms |
| HDIP | Hazardous Duty Incentive Pay |
| IAW | In Accordance With |
| IDA | Institute for Defense Analyses |
| ID | Identification |
| IDEF0 | Integrated Computer-Aided Manufacturing (ICAM) Definition Language 0 |
| IDEF1X | Integrated Computer-Aided Manufacturing (ICAM) Definition Language 1X |
| IR | Internal Review |
| IRA | Interface Requirements Agreements |
| IRR | Initial Requirements Review |
| LAN | Local Area Network |
| LID | Logical Identification (ID) |
| LL | Lessons Learned |
| LOGMOD-B | Logistics Module - Base level |
| LVM/OOD | Layered Virtual Machine/Object-Oriented Design |

| | |
|---|---|
| MAJCOM | Major Command (Air Force) |
| MDS | Manpower Data System |
| MPO | Military Pay Order |
| MSA | Modern Structured Analysis |
| NIST | National Institute of Standards and Technology |
| OMT | Object Modeling Technique |
| OO | Object-Oriented |
| OOS | Object-Oriented Specification |
| OOT | Object-Oriented Technology |
| ORD | Operational Requirements Document |
| OSE/IA | Open Systems Environment for Imminent Acquisitions |
| PC | Personal Computer |
| PDC | Public Domain Component |
| PMD | Program Management Directive |
| POSIX | Portable Operating System Interface for Computer Environments |
| PR | Problem Report |
| QT&E | Qualification Test and Evaluation |
| RA | Requirements Analysis |
| RCI | Remote Compilation Integrator |
| RDA | Remote Data Access |
| RDBMS | Relational Database Management System |
| RDF | Relational Design Facility |
| RPC | Remote Procedure Call |
| RTM | Requirements Tracking Matrix |
| SA | Structured Analysis |
| SCG | Software Control Group |
| SDD | Software Design Document |
| SDE | Standard Data Element |
| SDF | Software Development File |
| SDP | Software Development Plan |
| SDR | Software Design Review |
| SEER | System Evaluation and Estimation of Resources |
| SON | Statement of Operational Need |
| SOW | Statement of Work |
| SPS | Software Product Specification |

| | |
|---|---|
| SRR | Software Requirements Review |
| SRS | Software Requirements Specification |
| SSC | Standards System Center |
| SSDD | System/Segment Design Document |
| SSS | System/Segment Specification |
| STD | Software Test Description |
| STP | Software Test Plan |
| STR | Software Test Report |
| TAFIM | Technical Architecture Framework for Information Management |
| TBM | Theater Battle Management Group |
| TCP/IP | Transmission Control Protocol/Internet Protocol |
| TQM | Total Quality Management |
| TRM | Technical Reference Model |
| TRR | Test Readiness Review |
| USTRANSCOM | United States Transportation Command |
| VT | Virtual Terminal |